May 7–11, 2024
London, United Kingdom

Association for Computing Machinery

*Advancing Computing as a Science & Profession*


# ICPE Companion '24

## Companion of the 15th ACM/SPEC International Conference on Performance Engineering

*Sponsored by:*

***ACM SIGMETRICS, ACM SIGSOFT, & SPEC***

*General Chairs:*

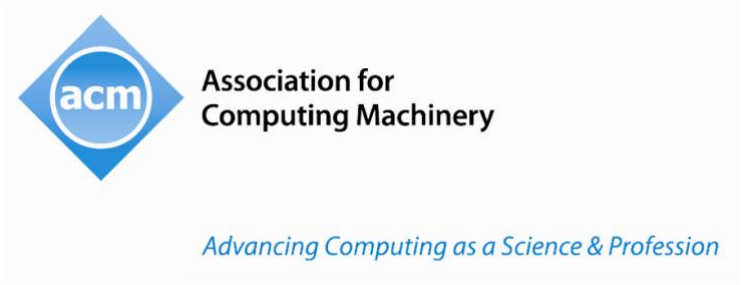
***Simonetta Balsamo (Ca' Foscari University of Venice, Italy)***
***William Knottenbelt (Imperial College London, UK)***

*Program Chairs:*

***Cristina L. Abad (Escuela Superior Politecnica del Litoral, Ecuador)***
***Weiyi Shang (University of Waterloo, Canada)***

*Publications Chairs:*

***Mauro Iacono (Università degli Studi della Campania Luigi Vanvitelli, Italy)***
***Jianing Qiu (Imperial College London, UK)***




**The Association for Computing Machinery**
**1601 Broadway, 10th Floor**
**New York, NY 10019-7434**

**Copyright © 2024 by the Association for Computing Machinery, Inc. (ACM).**
Permission to make digital or hard copies of portions of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission to republish from: permissions@acm.org or Fax +1 (212) 869-0481.

For other copying of articles that carry a code at the bottom of the first or last page, copying is permitted provided that the per-copy fee indicated in the code is paid through www.copyright.com.

**ISBN:** 979-8-4007-0445-1

Additional copies may be ordered prepaid from:

**ACM Order Department**
PO Box 30777
New York, NY 10087-0777, USA

Phone: 1-800-342-6626 (USA and Canada)
+1-212-626-0500 (Global)
Fax: +1-212-944-1318
E-mail: acmhelp@acm.org
Hours of Operation: 8:30 am – 4:30 pm ET

Printed in the USA

# ICPE 2024 Companion Welcome Message

It is our pleasure to welcome you to the *15th ACM/SPEC International Conference on Performance Engineering (ICPE)*, hosted at Imperial College London in South Kensington, London, UK, from May 7-11, 2024.

The International Conference on Performance Engineering (ICPE) is the leading international forum for presenting and discussing novel ideas, innovations, trends and experiences in the field of performance engineering.

This companion volume collects papers from the following ICPE 2024 tracks.

The *Emerging Research* track has the goal to present and discuss work-in-progress and new-ideas contributions, where some aspects of the work remain open. Seven papers were accepted in this track.

The *Artifacts* track received submissions for already accepted research track papers and three independently submitted data artifact papers that are presented in this volume. The Artifact Chairs oversaw a dedicated verification process with a team of reviewers willing to try out code, review datasets, and provide iterative feedback.

At its third successful edition, the *Data Challenge* track at ICPE asked participants to address challenges in analyzing datasets of microservices execution traces, as provided by the track organizers. Five papers were accepted in this track.

The *Posters and Demonstrations* track hosts three demo papers and two posters outlining work relevant to the performance engineering community.

The Tutorial track presents three high-quality tutorials on high-performance system design, industry perspectives on performance analysis, and distributed tracing analysis.

In addition to the main conference and its tracks, we were also very happy to host seven interesting colocated workshops on topics related to performance engineering.

These various tracks and workshops provide a valuable forum for discussing ideas in a broad range of topics related to performance engineering. They also provide a platform for researchers, students, and practitioners to discuss their ideas informally and establish collaborations.

ICPE 2024 will be held in person to stimulate the connection and socialization among young and senior researchers and practitioners. The organizing committee strongly believes that an "in-presence" conference is more effective in providing new insights and concrete opportunities to create strong relationships among ICPE community members.

As always, ICPE 2024 is the result of hard work made by the authors, the program committees of the different tracks and the external reviewers, and the conference organizers. We thank them for their invaluable contribution. We also acknowledge the individual workshop organizers and program committees for their contributions. Our sincere thanks go to SPEC and ACM, through SIGSOFT and SIGMETRICS, for their continuous support. We are also thankful to ACM SIGSOFT CAPS for providing travel grants to students and professionals, enabling increased participation.

On behalf of the whole organizing committee, we welcome you to South Kensington and hope you will enjoy the conference and the city of London.

**Simonetta Balasamo**
ICPE'24 General Co-Chair
Ca'Foscari University of Venice, Italy

**William Knottenbelt**
ICPE'24 General Co-Chair
Imperial College London, UK

*Chairs continued.*

**Cristina L. Abad**
ICPE'24 Program Co-Chair
Escuela Superior Politécnica del Litoral, Ecuador

**Vittoria de Nitto Personè**
ICPE'24 Emerging Research Co-Chair
Tor Vergata University of Rome, Italy

**Robert Ricci**
ICPE'24 Artifact Evaluation Co-Chair
University of Utah, USA

**Luca Traini**
ICPE'24 Data Challenge Co-Chair
University of L'Aquila, Italy

**André Bauer**
ICPE'24 Poster and Demo Co-Chair
University of Chicago, USA

**Heng Li**
ICPE'24 Tutorial Chair
Polytechnique Montreal, Canada

**Weiyi Shang**
ICPE'24 Program Co-Chair
University of Waterloo, Canada

**Lishan Yang**
ICPE'24 Emerging Research Co-Chair
George Mason University, USA

**Dmitry Duplyakin**
ICPE'24 Artifact Evaluation Co-Chair
NREL, USA

**Christoph Laaber**
ICPE'24 Data Challenge Co-Chair
Simula Research Laboratory, Norway

**Martin Straesser**
ICPE'24 Poster and Demo Co-Chair
University of Würzburg, Germany

# Welcome from the Workshops Chairs

It is our great pleasure to present the ICPE 2024 workshops program. ICPE workshops extend the main conference by providing a forum to foster discussion on hot and emerging topics from the broad field of performance engineering. They offer a highly dynamic venue to exchange ideas, establish new collaborations, and bootstrap debates on novel techniques, methodologies, and their associated early research results. Workshops feature various presentation formats, including research paper presentations, panel discussions, and keynote talks. Through these presentations and discussions with peer researchers, ICPE workshops help shape future research and identify promising research directions for performance engineering.

This year, the workshop program includes 7 workshops. Our program highlights both the continuity of well-established workshops and the emerging research directions of new workshops joining ICPE for the first time. These 7 workshops cover a notable range of topics from the perspective of performance engineering, including cloud computing, extreme-scale systems, and artificial intelligence.

The complete list of accepted workshops is:

- The Second International Workshop on Artificial Intelligence for Performance Modeling, Prediction, and Control (AIPerf 2024)
- The Fifth Workshop on Benchmarking in the Datacenter: Expanding to the Cloud (BID 2024)
- The Second Workshop on Serverless, Extreme-Scale, and Sustainable Graph Processing Systems (GraphSys 2024)
- The First Workshop on Performance Optimization in the LLM World (PerfLLM 2024)
- The Seventh Workshop on Hot Topics in Cloud Computing Performance (HotCloudPerf-2024)
- The Twelfth International Workshop on Load Testing and Benchmarking of Software Systems (LTB 2024)
- Ninth Workshop on Challenges in Performance Methods for Software Development (WOSP-C)

We thank all workshop chairs for organizing programs focusing on such diverse and inspiring topics, which attracted many high-quality paper submissions and enabled exciting workshop programs, featuring 30 accepted papers. Moreover, we would like to thank the technical program committee members of all workshops for their careful reviews. Further, we would like to thank the ICPE general chairs (William Knottenbelt and Simonetta Balsamo), the program chairs (Cristina L. Abad and Weiyi Shang), the publication chairs (Mauro Iacono and Jianing Qiu), the publicity and social-media chair (Marco Paolieri), and the web chair (Giordano d'Aloisio) for their help and support.

Finally, we thank all authors, keynote speakers, panelists, and anyone providing content that is so valuable for the workshops program. We are proud to welcome all ICPE workshop participants, and the community as a whole, to discuss and debate a broad range of topics for two full days. We are looking forward to meeting all of you at ICPE 2024, and we wish you a great experience attending the ICPE 2024 workshops!

**Diego Elias Costa**
ICPE 2024 Workshops Chair
Concordia University, Canada

**Michele Tucci**
ICPE 2024 Workshops Chair
University of L'Aquila, Italy

# Table of Contents

## Data Challenge Track



## Posters and Demonstrations Track



## Second International Workshop on Artificial Intelligence for Performance Modeling, Prediction, and Control (AIPerf 2024)



## Fifth Workshop on Benchmarking in the Datacenter: Expanding to the Cloud (BID 2024)



## Second Workshop on Serverless, Extreme-Scale, and Sustainable Graph Processing Systems (GraphSys 2024)

## First Workshop on Performance Optimization in the LLM World (PerfLLM 2024)

## Seventh Workshop on Hot Topics in Cloud Computing Performance (HotCloudPerf-2024)



## Twelfth International Workshop on Load Testing and Benchmarking of Software Systems (LTB 2024)

## Ninth Workshop on Challenges in Performance Methods for Software Development (WOSP-C)

# ICPE 2024: 15th ACM/SPEC International Conference on Performance Engineering Organization

**General Chairs:** Simonetta Balsamo (Ca' Foscari University of Venice, Italy)
William Knottenbelt (Imperial College London, UK)

**Program Chairs:** Cristina L. Abad (Escuela Superior Politecnica del Litoral, Ecuador)
Weiyi Shang (University of Waterloo, Canada)

**Journal-First Chairs:** Cristina L. Abad (Escuela Superior Politecnica del Litoral, Ecuador)
Weiyi Shang (University of Waterloo, Canada)

**Industry Track Chair:** Alexander Podelko (Amazon/AWS, USA)

**Emerging Research Track Chairs:** Vittoria de Nitto Personè (Tor Vergata University of Rome, Italy)
Lishan Yang (George Mason University, USA)

**Artifact Evaluation Chairs:** Robert Ricci (University of Utah, USA)
Dmitry Duplyakin (NREL, USA)

**Workshop Chairs:** Diego Costa (Concordia University, Canada)
Michele Tucci (University of L'Aquila, Italy)

**Tutorial Chair:** Heng Li (Polytechnique Montreal, Canada)

**Poster and Demos Chairs:** André Bauer (University of Chicago, USA)
Martin Straesser (University of Würzburg, Germany)

**Data Challenge Chairs:** Luca Traini (University of L'Aquila, Italy)
Christoph Laaber (Simula Research Laboratory, Oslo, Norway)

**Award Chairs:** Katinka Wolter (Freie Universitaet zu Berlin, Germany)
Marin Litoiu (York University, Canada)

**Publicity & Social-Media Chair:** Marco Paolieri (University of Southern California, USA)

**Finance Chair:** Tom Curtin (Imperial College London, UK)

**Publications Chairs:** Mauro Iacono (Università degli Studi della Campania Luigi Vanvitelli, Italy)
Jianing Qiu (Imperial College London, UK)

**Web Chair:** Giordano d'Aloisio (University of L'Aquila, Italy)

**Program Committee:** Antinisca Di Marco (University of L'Aquila)
Varsha Apte (Indian Institute of Technology – Bombay)
Yiming Tang (Rochester Institute of Technology)
Alexandru Iosup (VU)
Federica Sarro (University College London)
Murray Woodside (Carleton University)
Josef Spillner (Zurich University of Applied Sciences)
Philipp Leitner (Chalmers | University of Gothenburg)
Tingting Yu (University of Cincinnati)
Nikolas Herbst (University of Würzburg)
Cor-Paul Bezemer (University of Alberta)
Catia Trubiani (Gran Sasso Science Institute)
André van Hoorn (University of Hamburg)
Samuel Kounev (University of Wuerzburg)
Andre Bondi (Software Performance and Scalability Consulting LLC)
José Merseguer (Universidad de Zaragoza)
Valeria Cardellini (University of Roma "Tor Vergata")
Andrea Marin (Università Ca' Foscari Venezia)
Maria Carla Calzarossa (Universita di Pavia)
Raffaela Mirandola (Politecnico di Milano)
Steffen Becker (University of Stuttgart)
Marco Vieira (University of Coimbra)
Yintong Huo (Hong Kong University of Science and Technology)
Mauro Iacono (Università degli Studi della Campania "Luigi Vanvitelli")
Lishan Yang (George Mason University)
Alberto Avritzer (EsulabSolutions Inc.)
Wes Lloyd (University of Washington)
Junwen Yang (University of Chicago)
Petr Tuma (Charles University)
Patrick P. C. Lee (The Chinese University of Hong Kong)
Catalina M. Lladó (Universitat Illes Balears)
Sen He (The University of Arizona)
Vittorio Cortellessa (Università dell'Aquila)
Connie Smith (Performance Engineering Services)
Katinka Wolter (Freie Universitaet zu Berlin)
Jianmei Guo (East China Normal University)
Anne Koziolek (Karlsruhe Institute of Technology)
Wilhelm Hasselbring (Kiel University)
Evgenia Smirni (College of William and Mary)
Aris Leivadeas (École de technologie supérieure)
Shaohua Wang (Central University of Finance and Economics)
Daniele Di Pompeo (University of L'Aquila)

**Program Committee (continued):** Manoj Nambiar (Tata Consultancy Services)
Heng Li (Polytechnique Montréal)
André Bauer (University of Chicago)
Tse-Hsun (Peter) Chen (Concordia University)

**Program Committee (Industry track):** Muhammad Shoaib Bin Altaf (Oracle)
Vlastimil Babka (SUSE)
David Daly (MongoDB)
François Farquet (Oracle Labs)
Matt Flemming (Datastax)
Ajay Joshi (ARM)
Klaus-Dieter Lange (HPE)
Sai Sindhur Malleni (Red Hat)
Anoush Najarian (MathWorks)
Nishant Rawtani (HPE)
Daniel Seybold (benchANT)
Rekha Singhal (TCS)
Igor Trubin (Capital One)
Alexander Wert (Elastic)

**Program Committee (Artifact)** Aarushi Jain (University of Texas Arlington)
Aleksandra Kowalczuk (University of Warsaw)
Amit Samanta (University of Utah)
Elizabeth Ondula (University of Southern California)
Ghadeer Almusaddar (Binghamton University)
Hongyu Hè (ETH Zurich)
Jaiaid Mobin (Rochester Institute of Technology)
Juno Suárez (Portland State University)
Kevin Assogba (Rochester Institute of Technology)
Matthew Forshaw (Newcastle University)
Mazahir Hussain (Korea Institute of Science and Technology Information)
Muhammed Emin Ozturk (University of Utah)
Pegah Ahadian (Kent State University)
Rafael Herrera (University of Delaware)
Ryan Scherbarth (University of New Mexico)
Saheed Olayemi Bolarinwa (The Leibniz Supercomputing Centre)
Sören Henning (Johannes Kepler University Linz)
Stephen Nicholas Swatman (University of Amsterdam)
Swetha Varadarajan
Tamoghna Sarkar (University of Southern California)
Souptik Sen (Snowflake)
Urjoshi Sinha (Lawrence Berkeley National Lab)
Radhakrishnan Venkataramani (Snowflake)

**Program Committee (Data Challenge):**
Jinfu Chen (Wuhan University)
Zishuo Ding (University of Waterloo)
Martin Grambow (Technische Universität Berlin)
Sen He (University of Arizona)
Lizhi Liao (University of Waterloo)
Max Weber (Leipzig University)
Chenxi Zhang (Fudan University)

**Additional Reviewers:**
Andres Abad
Palak Bhandari
Prateek Bhatnagar
Andrea Bianchi
Vanessa Borst
Sophie Corallo
Robert Cordingly
Timo Dittus
Daniel Grillmeyer
Marius Hadry
Zhihan Jiang
Supriya Kamthania
Shruti Kunde
Yichen Li
Chuansheng Lu
Yannik Lubas
Maximilian Meissner
Diletta Olliaro
Kebin Peng
Ivo Rohwer
Larissa Schmid
Seyedehhaleh Seyeddizaji
Shuai Shao
Anushree Singh
Michael Stenger
Rajesh Tadakamadla
Chen Zou

# ACM/SPEC ICPE 2024 Sponsors & Supporters

**Sponsors:**




**Supporters:**



# Context-aware Root Cause Localization in Distributed Traces Using Social Network Analysis (Work In Progress paper)

Mahsa Panahandeh
Electrical and Computer Engineering department,
University of Alberta
Edmonton, Alberta, Canada
panahand@ualberta.ca

Naser Ezzati-Jivan
Department of Computer Science, Brock University
St. Catharines, Ontario, Canada
nezzatijivan@brocku.ca

Abdelwahab Hamou-Lhadj
Department of Electrical and Computer Engineering,
Concordia University
Montreal, Quebec, Canada
wahab.hamou-lhadj@concordia.ca

James Miller
Department of Electrical and Computer Engineering,
University of Alberta, Canada
Edmonton, Alberta, Canada
jimm@ualberta.ca

## ABSTRACT

The complexity of microservices and their distributed nature necessitates constant monitoring and tracing of their execution to identify performance problems and underlying root causes. However, the large volume of collected data and the complexity of distributed communications pose challenges in identifying and locating abnormal services. In this paper, we propose a novel approach that takes into consideration the importance of execution contexts in propagating and localizing performance root causes. We achieve this by integrating social network analysis techniques with spectrum analysis. To evaluate our proposed approach, we conducted an experiment using a real-world benchmark, and we observed promising preliminary results, with a success rate of 91.3% in correctly identifying the primary root cause (top-1), and a perfect 100% success rate in finding the root cause within the top three candidates (top-3).

## CCS CONCEPTS

• **Software and its engineering → Software reliability**; **Software performance**.

## KEYWORDS

Root-cause Localization, Social Network Analysis, Spectrum Analysis, Distributed Traces, Contextual Analysis

**ACM Reference Format:**

Mahsa Panahandeh, Naser Ezzati-Jivan, Abdelwahab Hamou-Lhadj, and James Miller. 2024. Context-aware Root Cause Localization in Distributed Traces Using Social Network Analysis (Work In Progress paper). In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3629527.3651426

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651426

## 1 INTRODUCTION

Despite the widespread adoption of microservices for their scalability, modularity, and rapid deployment capabilities, their distributed architecture introduces significant challenges in diagnosing performance issues and localizing their root causes. Consider a complex e-commerce platform built upon this architecture. Services frequently depend on each other to accomplish tasks. When a performance issue such as a slowdown occurs, it rarely remains isolated but instead propagates through dependent services. A performance issue, say a slowdown, in a single service can have a cascading effect on all services dependent on it. Alternatively, it might also be the case that poor performance in a particular service is actually rooted in another service it depends upon. Without understanding these structural interconnected dependencies, diagnosing issues will become a complicated process. In the worst cases, the actual root cause may be entirely overlooked.

Therefore, accurate diagnosis requires understanding services interactions and dependencies. This necessitates an in-depth study of structural dependencies, particularly in cases of forward or backward anomaly propagation where the starting point of an issue might be several services away from where it eventually manifests.

Understanding complex inter-dependencies is essential for comprehending anomaly propagation and troubleshooting distributed systems, a topic explored in various research works [5, 8, 13, 15]. Some researchers advocate for prioritized diagnostic processes based on the likelihood of anomaly propagation in different components [4, 18]. However, the specific challenges faced by existing studies vary. Certain studies [13] face challenges in precisely pinpointing the direction of dependencies and analyzing the propagation path of anomalies. Others are restricted to insights derived solely from abnormal system executions, overlooking the information provided by normal request propagation [8]. Some studies [4, 5, 15, 18] remain constrained, often relying on isolated aspects of individual services in anomaly propagation, lacking the holistic view needed for accurate root cause identification. This limitation stems from the inherent structural complexity of distributed systems, where services are interconnected and interdependent, adding multiple layers of complexity to anomaly resolution.

To address existing limitations, we introduce a strategy using social network algorithms to enhance spectrum analysis, a method for estimating system component faultiness based on successful and failed executions. Our approach incorporates three key contexts: individual services, service communities, and execution paths. Then, it explains how anomalies propagate through these contexts and identifies key nodes where performance issues likely originate. Utilizing graph theory concepts like PageRank and community detection adds a new layer of depth to our spectrum analysis. The result is a context-aware, finely-grained method for accurately pinpointing the root causes of performance issues. Our approach utilizes distributed traces [11], to represent these contexts.

Upon detecting anomalies, we generate graphs and use social network techniques to calculate the structural importance of services, including an assessment of execution path importance. The resulting impact scores are used as weights in our spectrum analysis, generating a ranked list of probable root cause services, thus reducing the debugging effort required by programmers [17].

Our preliminary tests on an instance of a real-world production microservice system in China Mobile Zhejiang, a known complex and large-scale system, indicate a 91.3% success rate of our approach in identifying the primary root cause (top-1) and a 100% success rate for locating the root cause among the top three candidates (top-3) across various scenarios. Our preliminary tests employed rigorous sampling and evaluation metrics, ensuring the robustness of our findings. We have designed our method for generalizability, so it can be applied in many different situations and we have concrete plans for future empirical validation to strengthen our findings.

Our main contributions are as follows: I) Employing social network analysis to construct and analyse the structural interdependence of services, communities, and execution paths for forward and backward anomaly propagation and root cause identification, II) Proposing an enhanced weighted spectrum analysis. While existing methods often rely on individual services and isolated executions, our approach breaks new ground by introducing contextual layers, encompassing individual services, service communities, and execution paths, into root cause analysis. This context-aware methodology refines spectrum analysis, providing a more accurate identification of root causes, and thus advances the state-of-the-art.

## 2 FOUNDATIONAL EXPERIMENTS

Spectrum-based techniques are commonly used for debugging and fault localization in software applications. These techniques gather various types of test coverage data to identify likely root causes of failures. Specifically, they collect metrics such as $O_{ef}$, $O_{nf}$, $O_{ep}$, and $O_{np}$, which count the presence or absence of a given component in both failed and successful test cases [18]. A risk factor, like the Ochiai risk factor, is then used to quantify the suspicion level for each component being the root cause [3, 10]. In microservices, spectrum-based methods utilize distributed traces for both normal and abnormal system states to perform similar analyses [17].

Applying spectrum-based methods to distributed traces has limitations in root cause localization. For instance, our experiments, shown in Figure 1, based on a scenario from dataset C published by Li et al. [6], found that the original spectrum analysis ranked the true root cause (docker_003) only fifth in suspicion, while it placed os_021 at the top, identified as the most suspicious due to having the highest Ochiai score of 0.319. Yu et al.'s approach [18], which integrates a personalized PageRank algorithm into spectrum analysis, also falls short. It places emphasis on service frequency in determining the significance of an execution path for root cause localization, but it lacks granularity in understanding anomaly propagation, leading to suboptimal root cause identification. Figure 1, middle table, shows our experiments when we integrate a PageRank algorithm with spectrum analysis. It identifies the true root cause in the third place, while os_021 remains at the top.

In our experiments, we noticed two issues with integrating spectrum analysis with PageRank for root cause identification. First, about 30% of the cases presented multiple services with identical suspicion scores, complicating the ranking. Second, both original and integrated methods struggle when the root cause does not frequently appear in abnormal traces, e.g., the presence of a frequent loop between non-true root cause services in abnormal traces.

To address these issues, we study the significance of services in interconnected groups (communities) rather than prioritizing them based on their frequency across all traces. Here, by 'communities,' we refer to clusters of services that frequently interact with each other, thereby forming a closely-knit functional group within the larger system. Finding communities assists in distinguishing observed contexts and differentiating between similar connectivity patterns, which highly reduces (up to 98%, according to our preliminary experimental results) the likelihood of encountering multiple services with the same suspicious score all occupying the same position in the rank list of candidates. Additionally, studying the significance of services according to the significance of their interconnected services prioritizes less frequently observed services when they are significant within their own context. Therefore, the root cause service can be detected even if it is not frequently invoked in the collected abnormal traces. In addition to studying the significance of services in communities, we introduce a novel aspect to the root cause identification process by evaluating contextual details in requests or traces. While existing research by Yu et al. [18] assigns more weight to shorter traces, our approach innovatively refines this by equalizing the importance of services and trace diversity, regardless of the trace length. This offers a more comprehensive and effective method for root cause localization. Please refer to the right table in Figure 1 for a comparison between our method (context-aware root cause localization) and traditional approaches, which clearly illustrates the efficacy of our strategy. With our approach, the true root cause is ranked at the top with an Ochiai score of 0.286.

## 3 SYSTEM DESIGN

Figure. 2 represents our context-aware root cause localization approach including several steps of data collection, Service Call Graph (SCG) construction, social network analysis, and spectrum analysis.

### 3.1 Data Collection

Our process is initiated in response to a detected anomaly. Once an anomaly is detected, we collect normal and abnormal distributed traces within a specified time window (5 minutes in this paper). Any

| Services | Original Spectrum Analysis [15] | | | | | Spectrum Analysis Integrated with PageRank | | | | | Context-aware Root Cause Localization | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $O_{ef}$ | $O_{nf}$ | $O_{ep}$ | $O_{np}$ | Ochiai score | Weighted $O_{ef}$ | Weighted $O_{nf}$ | Weighted $O_{ep}$ | Weighted $O_{np}$ | Ochiai score | Weighted $O_{ef}$ | Weighted $O_{nf}$ | Weighted $O_{ep}$ | Weighted $O_{np}$ | Ochiai score |
| docker_007 | 0 | 115 | 208 | 240 | 0 | 0 | 0 | 10.85 | 12.52 | 0 | 0 | 0 | 106.51 | 0 | 0 |
| os_023 | 0 | 115 | 1 | 447 | 0 | 0 | 0 | 0.28 | 12.79 | 0 | 0 | 0 | 0.075 | 33.52 | 0 |
| os_021 | 59 | 56 | 237 | 211 | 0.319 | 6.11 | 5.80 | 14.87 | 13.24 | 0.386 | 13.67 | 12.98 | 237.0 | 211.0 | 0.167 |
| db_003 | 0 | 115 | 448 | 0 | 0 | 0 | 0 | 59.99 | 0 | 0 | 0 | 0 | 184.70 | 0 | 0 |
| docker_006 | 0 | 115 | 214 | 234 | 0 | 0 | 0 | 10.87 | 11.88 | 0 | 0 | 0 | 31.45 | 34.39 | 0 |
| os_022 | 56 | 59 | 229 | 219 | 0.309 | 3.57 | 3.76 | 25.82 | 24.69 | 0.243 | 18.32 | 19.30 | 211.82 | 202.57 | 0.196 |
| docker_003 | 29 | 86 | 113 | 335 | 0.226 | 6.95 | 20.61 | 7.55 | 22.39 | 0.347 | 29.0 | 86.0 | 60.38 | 179.02 | 0.286 |
| docker_004 | 30 | 85 | 110 | 338 | 0.236 | 7.29 | 20.65 | 7.30 | 22.47 | 0.361 | 23.04 | 65.29 | 110.0 | 338.0 | 0.212 |
| docker_001 | 33 | 82 | 114 | 334 | 0.253 | 4.11 | 10.22 | 9.40 | 27.55 | 0.295 | 18.56 | 46.13 | 86.89 | 254.59 | 0.224 |
| db_007 | 0 | 115 | 448 | 0 | 0 | 0 | 0 | 18.62 | 0 | 0 | 0 | 0 | 30.63 | 35.34 | 0 |
| docker_005 | 3 | 112 | 320 | 128 | 0.015 | 0.08 | 3.11 | 15.28 | 6.11 | 0.011 | 0.33 | 12.34 | 46.99 | 18.79 | 0.013 |
| docker_008 | 0 | 115 | 216 | 232 | 0 | 0 | 0 | 11.41 | 12.25 | 0 | 0 | 0 | 31.66 | 34.0 | 0 |
| db_009 | 0 | 115 | 448 | 0 | 0 | 0 | 0 | 53.47 | 0 | 0 | 0 | 0 | 208.58 | 0 | 0 |
| docker_002 | 25 | 90 | 111 | 337 | 0.199 | 4.93 | 17.75 | 9.05 | 27.50 | 0.276 | 25.0 | 90.0 | 11.0 | 337.0 | 0.199 |

**Figure 1: Comparison of Original, PageRank-Integrated, and Context-aware Weighted Spectrum Analyses in a Real-world Scenario (True root cause: docker_003, Top-1 identified root cause by each method has been highlighted.)**

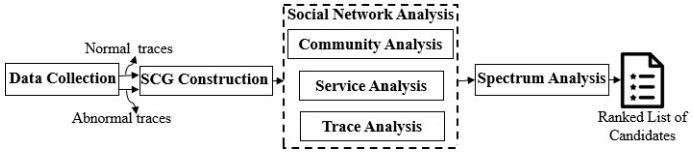



**Figure 2: Context_aware Root Cause Localization**

state-of-the-art anomaly detection techniques [12] can be employed to detect performance anomalies and label distributed traces as normal or abnormal. In this paper, we utilize the anomaly detection approach proposed by Li et al. [4]. However, as recommended by the literature [18], subsequent occurrences of the same anomalous state within the time window are not treated as separate anomalies.

## 3.2 SCG Construction

Following the data collection phase, two SCGs, built from each group of normal and abnormal traces, serve as weighted graphs. In these graphs, nodes symbolize services, edges represent service calls, and edge weights quantify the frequency of these calls within the respective set of traces. Distributed traces provide the necessary context information, capturing parent-child relations between services, which helps in constructing SCGs.

## 3.3 Social Network Analysis

In this phase, social network methods are employed to assess the structural influence of services communities, individual services, and traces during anomalies within SCGs. The output comprises importance scores for services and traces in both normal and abnormal conditions.

*3.3.1 Community Analysis.* Upon constructing normal and abnormal SCGs, we apply the Louvain graph community algorithm [1] to each SCG to partition them into smaller, closely related contextual communities. This aids in identifying cohesive groups and strong inter-node communication. For example, in abnormal SCG, these communities highlight partitions susceptible to anomalies should they contain an anomaly-affected node.

The Louvain method has two phases. First, nodes within an SCG are iteratively assessed and assigned to neighbouring nodes based on modularity cost function gains[1], continuing until no more modularity gains are achievable. Second, communities identified in the first phase are amalgamated into supervertices, converting nodes within each community into a single node. Supervertices' connectivity depends on at least one edge between nodes from corresponding communities, with the edge weight determined by the sum of all edges weight between their respective lower-level partitions. The algorithm iteratively applies these phases to supergraphs until communities stabilize, typically after a few rounds.

*3.3.2 Service Analysis.* Next, after identifying the community contexts, we quantify services importance within their community. To this end, we adopt the suggested approach in trace abstraction by Wang et al. [14]. First, we use an iterative PageRank algorithm [16] for each community to determine the significance of services based on their interactions within the community. The process begins by initializing the PageRank values for each service within the community. These initial values are set to $\frac{1}{n}$, where $n$ represents the total number of services in the community. Subsequently, the PageRank values are iteratively calculated until they converge to a stable value. The PageRank for a service $n$ in a community of $C$ is determined based on the PageRanks of its neighbouring services over $t$ iterations, and it can be defined as follows:

$$PR(n)_t = \alpha \sum_{(n \to n') \in \text{ edges of } C} \frac{PR_{t-1}(n')}{out\_degree(n')} \quad (1)$$

where $\alpha$ is a normalization factor for the total rank of all services and out_degree is the number of outgoing edges from $n'$.

This step is performed for services in identified communities of both normal and abnormal SCGs.

*3.3.3 Trace Analysis.* In this step, we focus on prioritizing different request types considering their effectiveness in uncovering a root

cause. According to Yu et al. [18], less diverse traces expedite root-cause localization. This is because more similar traces indicate a narrower scope of difference, simplifying the pinpointing of the root cause. However, Yu et al. measure trace diversity using the count of operations covered in traces, which makes it dependent on trace size. Consequently, shorter traces with less important services may overshadow longer ones with more critical services. In our approach, we prioritize traces by considering both their diversity and the significance of the services they cover, regardless of the number of services involved.

We first, cluster collected traces based on their request type separately for normal and abnormal distributed traces. This approach ensures that we study all observed request types, regardless of how frequently they have occurred. Then, for each request-type cluster, we calculate a score based on the diversity and importance of covered services.

To measure the importance score of clusters based on the importance of the services they cover, we adjust the formula recommended by Chen et al. [2] as follows. This refinement allows us to measure the rank score of a cluster $cl_i$ based on the PageRank score (PR) of services they cover. In this context, PR is the computed scores using equation 1, The function $L(X)$ denotes the position of score $X$ within the ordered list of ascending PR scores within the SCG, and |SCG| represents the number of SCG's nodes (services). For abnormal request-type clusters, SCG refers to abnormal SCG and PRs are scores of all services computed from abnormal SCG. The same is applied to normal request-type clusters.

$$Rank(cl_i) = \frac{L(Max(PRs \in cl_i) - 1}{|SCG|} + \frac{Mean(PRs \in cl_i)}{|SCG|sum(PRs \in SCG)} \quad (2)$$

Next, we measure diversity between request-type clusters using Jaccard distance [7], favouring less diverse clusters. Finally, the adapted heuristic search algorithm [2] searches for the next cluster based on the prior one, aiming to maximise the sum of Rank($cl_i$) score while minimizing the cluster diversity.

## 3.4 Spectrum Analysis

Considering the importance of services in the community context and request type (trace), we redefine spectrums. For instance, $O_{ef}$ is modified as follows where, T is a set of abnormal traces of $T_1, T_2, ..T_k$ including service $s_i$, $cl_{T_j}$ is the abnormal request-type cluster for trace $T_j$, and $PR(s_i)$ is the Pagerank for $s_i$ in the abnormal SCG.

$$O_{ef}(s_i) = PR(s_i) \times \sum_{\forall T_j \ including \ s_i, T_j \in T} rank(cl_{T_j}) \quad (3)$$

Similarly, other notation definitions are updated by being influenced by the rank scores, while $O_{ep}$ and $O_{np}$ are computed based on the normal SCG, normal set of traces, and normal request-type clusters identified in the previous step.

To estimate the suspicious score using the notations, we use the Ochiai factor [10], measured for each service $s_i$ as:

$$Ochia(s_i) = \frac{O_{ef}}{\sqrt{(O_{ef} + O_{ep})(O_{ef} + O_{nf})}} \quad (4)$$

# 4 PRELIMINARY RESULT AND DISCUSSION

We evaluate the efficiency of our approach by conducting experiments on a real-world microservice benchmark (dataset C), provided during the 2020 AIOps Challenge Event[1] [6]. Given that this dataset extends beyond microservice applications, in alignment with literature recommendations [4, 18], we exclusively focus on faults related to microservice. Our evaluation involves the random selection of 46 time windows, each containing labelled root causes, as well as corresponding normal and abnormal distributed traces. Our experiment dataset includes 15 instances of CPU stress, 15 cases of network delays, and 16 occurrences of network loss. We define "Top-1" to "Top-3" as the probability of locating the true root causes within the top 1 to 3 service instances among all services, descendingly sorted based on their computed Ochiai score. This sorted list of ranked services is referred to as the ranked list of candidates [8, 18]. Figure. 3 shows the result of root cause identification by our context-aware root cause localization approach for all 46 time windows compared to the original spectrum analysis [17] and spectrum analysis integrated with PageRank, inspired by [18]. The context-aware root cause localization outperforms the spectrum analysis integrated with PageRank by 13.5% in detecting the true root cause at the top position of the ranked list of candidates, and it performs 35% better than the original spectrum analysis.

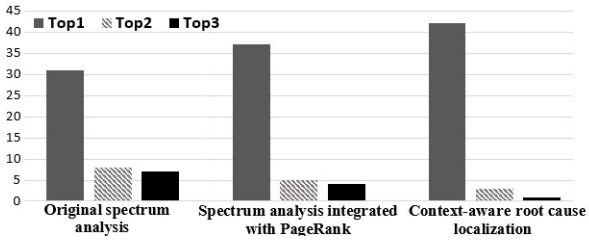



**Figure 3: Performance Comparison Across 46 Scenarios: Original Spectrum Analysis [17] vs. Enhanced Methods**

Our approach also exhibits a higher success rate in detecting root causes of CPU exhaustion scenarios, followed by network loss scenarios, and finally network delay cases. This performance variation can be attributed to the ability of our approach to effectively capture abnormal behavioural patterns within the collected traces. As anomalies propagate through more traces and spans, they become more likely to be successfully identified. Our initial investigation revealed that CPU exhaustion affected a larger number of services compared to the delay. This discrepancy can be attributed to the different approaches used to inject these anomalies into the system.

To evaluate the importance of studying services in contexts of communities and traces for root cause localization, we examine a scenario containing more than 530 traces collected after injecting the CPU stress into one of the services of the benchmark. After labelling normal and abnormal traces identified by the anomaly detection stage, we perform an original spectrum analysis to find the root cause. We then incorporate our approach components into the original spectrum analysis one by one to highlight how each contributes to enhancing the result of root cause localization.

[1] https://github.com/NetManAIOps/AIOps-Challenge-2020-Data

Table 1 demonstrates the result of each improvement applied to the same scenario, indicating the position within the ranked list of candidates where the root cause was identified by that specific improvement. The original spectrum analysis [17] locates the underlying root cause of this scenario in the sixth position of the ranked list of candidates. As shown, using our context-aware root cause localization, the true root cause is identified at the top position while each component also enhances the accuracy of root cause identification.

**Table 1: Performance Ranking of Localization Components**

| Method | Pos. |
|---|---|
| Orig. Spectrum [17] | 6th |
| Orig. + Service PageRank | 4th |
| Orig. + Community_based Service PageRank | 2nd |
| Context-aware: Full Method | 1st |

## 5 RELATED WORK

Root cause localization has become prominent in recent research due to its significant role in assuring the quality of complex systems [9, 13, 15, 17, 18]. A common approach to finding root causes involves studying the dependencies between services or traces to understand anomaly propagation, ultimately pinpointing the underlying root cause [5, 8, 15]. However, certain research works [13], face limitations in determining the propagation direction between dependent components. Furthermore, a significant portion of root-cause localization methods concentrates solely on abnormal executions [8, 13, 15].

Ye et al. [17] emphasize the significance of using both normal and abnormal traces, proposing a root cause localization approach based on an original spectrum analysis that leverages all traces to identify root causes. Addressing the requirements of spectrum analysis for distributed traces, Yu et al. [18] suggest integrating a personalized PageRank algorithm with the original spectrum analysis. The proposed personalized PageRank prioritizes services and traces based on their importance in uncovering root causes. However, as mentioned in Section 2, we found that studying the importance of individual services in Yu et al. study [18], is not always effective, especially when the true root cause is not frequently observed in abnormal traces. Moreover, Yu et al. [18] study the importance of traces in revealing the root cause based on their frequency and length, which is not always applicable, especially when the true root cause occurs in longer traces calling only a few services. These limitations are also discussed in Sections 2 and 3.

To address these limitations, we incorporate spectrum analysis with social network concepts to assess structural importance in forward and backward anomaly propagation across interconnected services. Studying the importance of services in interconnected communities helps analyze the significance of services within their respective communities. This approach overcomes biased rankings of services based solely on their high outgoing connections, neglecting the density of connections in overall SCGs. Moreover, to examine the importance of trace scope in uncovering root causes, we introduce a heuristic search algorithm to simultaneously evaluate trace importance based on both the significance and diversity of covered services within the trace scope. This makes trace scope analysis independent of the trace length and differentiates traces based on what they cover rather than their length. As our preliminary results show, this improves upon simply counting called services in each trace context, as done in Yu et al.'s work [18].

While there are research works employing social network analysis in root cause localization, they are often limited in utilizing techniques for studying the prominence of individual nodes within a network [9, 18, 19]. To the best of our knowledge, our work stands as the only research investigating the impact of different levels of contextual structures, such as individual services, service communities, and traces, in uncovering root cause localization using social network techniques.

## 6 CONCLUSIONS AND FUTURE PLAN

This study introduces a novel context-aware approach for root cause localization in distributed systems. Our methodology underscores the pivotal role of service communities, individual services, and trace scope in pinpointing the root causes of system anomalies. Through our work, we have effectively mitigated the issues delineated in Section 2. Preliminary outcomes showcase a high success rate, ranging from 91.36% to 100%, in accurately identifying root causes across diverse settings.

Moving forward, there are several directions for further improvement and extension. Firstly, we plan to explore modifications to obtain more detailed models beyond SCGs. These models can incorporate additional modalities, such as profiling metrics, enabling us to add performance insights to our social network analysis and analyze execution states instead of solid services. This will also help provide explanations about the issues associated with the ranked candidates, aiding in debugging or further investigations.

Furthermore, we aim to investigate how our approach impacts the detection of multi-root causes within service communities. By focusing on the interplay of anomaly propagation within communities, we anticipate that our methodology may excel in identifying complex scenarios where multiple root causes manifest within or across service communities.

Additionally, we aim to explore and adapt network analysis concepts that align with the unique characteristics of distributed traces. By leveraging these concepts, we can further investigate their correlation with different system types or collected data, potentially uncovering new insights instrumental in customizing network analysis concepts, such as community detection for distributed traces, thereby enhancing the precision of our methodology.

In conjunction with these efforts, we plan to evaluate our approach across a diverse set of case studies varying in size, complexity, and nature of their design. This assessment will help us gauge the scalability and adaptability of our approach for real-world systems.

Lastly, we acknowledge the importance of performing comprehensive comparative assessments against established methods. Such evaluations are crucial for validating the efficacy of our approach and identifying its advantages, shortcomings, and avenues for refinement across different use cases.

# REFERENCES


[1] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[2] Lizhe Chen, Ji Wu, Haiyan Yang, and Kui Zhang. 2022. Does PageRank apply to service ranking in microservice regression testing? *Software Quality Journal* 30, 3 (2022), 757–779.

[3] James A. Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of Test Information to Assist Fault Localization. In *Proceedings of the 24th International Conference on Software Engineering* (Orlando, Florida) *(ICSE '02)*. Association for Computing Machinery, New York, NY, USA, 467–477. https://doi.org/10.1145/581339.581397

[4] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, Zhekang Chen, Wenchi Zhang, Xiaohui Nie, Kaixin Sui, and Dan Pei. 2021. Practical Root Cause Localization for Microservice Systems via Trace Analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. 1–10. https://doi.org/10.1109/IWQOS52092.2021.9521340

[5] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, et al. 2022. Actionable and interpretable fault localization for recurring failures in online service systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 996–1008.

[6] Zeyan Li, Nengwen Zhao, Shenglin Zhang, Yongqian Sun, Pengfei Chen, Xidao Wen, Minghua Ma, and Dan Pei. 2022. Constructing large-scale real-world benchmark datasets for AIOps. *arXiv preprint arXiv:2208.03938* (2022).

[7] Jackson A Prado Lima and Silvia R Vergilio. 2020. Test Case Prioritization in Continuous Integration environments: A systematic mapping study. *Information and Software Technology* 121 (2020), 106268.

[8] JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*. Springer, 3–20.

[9] Leonardo Mariani, Cristina Monni, Mauro Pezzé, Oliviero Riganelli, and Rui Xin. 2018. Localizing faults in cloud systems. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 262–273.

[10] Lee Naish, Hua Jie Lee, and Kotagiri Ramamohanarao. 2011. A model for spectra-based software diagnosis. *ACM Transactions on software engineering and methodology (TOSEM)* 20, 3 (2011), 1–32.

[11] Austin Parker, Daniel Spoonhower, Jonathan Mace, Ben Sigelman, and Rebecca Isaacs. 2020. *Distributed tracing in practice: Instrumenting, analyzing, and debugging microservices*. O'Reilly Media.

[12] Jacopo Soldani and Antonio Brogi. 2022. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)* 55, 3 (2022), 1–39.

[13] Jörg Thalheim, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. 2017. Sieve: Actionable insights from monitored metrics in distributed systems. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. 14–27.

[14] Ji Wang and Naser Ezzati-Jivan. 2020. Enhanced execution trace abstraction approach using social network analysis methods. *Softwaretechnik-Trends* 40, 3 (2020), 58–60.

[15] Li Wu, Johan Tordsson, Jasmin Bogatinovski, Erik Elmroth, and Odej Kao. 2021. MicroDiag: Fine-grained Performance Diagnosis for Microservice Systems. In *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*. 31–36. https://doi.org/10.1109/CloudIntelligence52565.2021.00015

[16] W. Xing and A. Ghorbani. 2004. Weighted PageRank algorithm. In *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.* 305–314. https://doi.org/10.1109/DNSR.2004.1344743

[17] Zihao Ye, Pengfei Chen, and Guangba Yu. 2021. T-Rank:A Lightweight Spectrum based Fault Localization Approach for Microservice Systems. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 416–425. https://doi.org/10.1109/CCGrid51090.2021.00051

[18] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 3087–3098. https://doi.org/10.1145/3442381.3449905

[19] Guangba Yu, Zicheng Huang, and Pengfei Chen. 2021. TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. *Journal of Software: Evolution and Process* (2021), e2413.

# Enhancing the Performance of Deep Learning Model Based Object Detection using Parallel Processing (Work In Progress Paper)

Omar Imran
Carleton University
Ottawa, ON, Canada
omarimran@cmail.carleton.ca

Shikharesh Majumdar
Carleton University
Ottawa, ON, Canada
majumdar@sce.carleton.ca

Sreeraman Rajan
Carleton University
Ottawa, ON, Canada
sreeramanr@sce.carleton.ca

## ABSTRACT

The need for accelerated object detection is paramount for safety critical applications such as autonomous vehicles. This paper focuses on leveraging parallel processing techniques for enhancing the performance of object detection. Specifically, this research engineers system performance by timely detection of common objects encountered by vehicles, such as other automobiles, pedestrians, and bicycles. Deploying popular pretrained deep learning models like the You Only Look Once (YOLO) model within the Apache Spark framework, the potential enhancements in detection speed achieved through parallel processing are investigated. The capability of the system to efficiently handle large datasets and distribute time-critical applications across multiple nodes is explored to improve both latency and scalability. The one-factor-at-a-time method is used to assess the impact of different system and workload parameters on performance. Of particular interest is the impact of Spark data partitioning on performance, especially for driving scenarios where the number of objects are changing rapidly. A novel data partitioning technique that uses the principles of entropy is utilized. The overall performance objective of this research will be to improve speed for object detection in cars which can improve safety in time critical events such as sudden braking or turning.

## CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**; **Object detection**.

## KEYWORDS

Parallel Processing, Object Detection, Deep Learning, Apache Spark, Video Processing

**ACM Reference Format:**
Omar Imran, Shikharesh Majumdar, and Sreeraman Rajan. 2024. Enhancing the Performance of Deep Learning Model Based Object Detection using Parallel Processing (Work In Progress Paper). In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651427

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651427

## 1 INTRODUCTION

Timely detection of objects is crucial in safety critical applications such as autonomous vehicles and has become an important subject of research due to increase of accidents happening with autonomous vehicles such as self driving cars [10, 21, 23]. Particularly, the incident when a Tesla in autopilot mode failed to stop at a red light and crashed it into another car killing two people questioned the safety of autonomous vehicles [24]. Object detection is crucial for ensuring the safety of autonomous vehicles and other safety critical applications, such as the detection of objects by Unmanned Aerial Vehicles [32]. The focus of this paper is to use parallel processing to improve object detection.

This work in progress paper specifically concentrates on the identification of objects like pedestrians, traffic lights, and trucks encountered by moving vehicles. The research will utilize the MIT DriveSeg dataset, featuring video frames captured by a camera mounted on a moving vehicle [6]. These video frames capture various driving environments including rural roads, busy highways, and city streets, enabling the simulation of diverse scenarios encountered by vehicles in daily life. Then, using the You Only Look Once Version 3 (YOLOv3) pretrained deep learning model and the Apache Spark parallel computing framework, object detection on the frames will be carried out in parallel. This research will utilize Amazon Web Services (AWS) cloud servers to create scalable Spark clusters which will facilitate in the deployment of the proof-of-concept prototype used in the different experiments. The experiments will use the one-factor-at-a-time approach [27] to systematically alter various system parameters, including the number of worker nodes and the average number of objects in each scenario, to assess their individual impact. This paper will also explore the effect of data partitioning strategies on performance when evenly distributing dynamically changing frames across the various nodes. The paper will introduce a unique method for estimating workload and allocating frames across partitions using frame entropy.

The contributions of this paper include:

- A frame entropy based novel technique for workload estimation and frame allocation across the different nodes in a parallel processing platform such as Spark.
- A proof-of-concept prototype deployed on Spark for analyzing the performance of the concurrent object detection techniques discussed in the paper. This includes combining the YOLOv3 pretrained deep learning model with Apache

Spark's distributed computing framework thus enabling the performance engineering of deep learning based object detection on parallel platforms.
- Initial insights into the impact of different system and workload parameters on the performance of the object detection techniques.

Further contributions to the state of the art are expected from continuation of the research as discussed in Section 6.

Previous research done has shown a 60% performance increase when deploying deep learning models on Spark clusters, suggesting the potential of parallel programming in object detection tasks [20]. The YOLOv3 model has been vastly used in the literature for object detection for applications such as the detection of unmanned aerial vehicles [11]. Furthermore, research has been done in the literature as well on different partitioning algorithms in Spark. They analyzed different partitioning approaches for a textual dataframe and concluded that Spark's standard random Hash Partitioning could be optimized further with custom partitioning strategies [22]. This further indicates that the choice of partitioning algorithm could also play a significant role in other applications such as enhancing performance in video processing. Notably, while most research papers emphasize the quantity of data points in partitioning studies, this paper uniquely concentrates on the specific content within each partition to estimate workload.

This paper is organized as follows. Section 2 gives a background on the methodologies used in this research. Section 3 gives an overview on the design of the system and the experiments. Section 4 goes over the experimental results. Section 5 concludes the paper and Section 6 gives a summary of the steps for further work.

# 2 BACKGROUND

This section will provide a background on the different topics used in this paper.

## 2.1 Apache Spark

Apache Spark can be used to efficiently distribute workloads and data across a cluster of machines or nodes and is used for distributing the processing of the video frames. This distributed computing framework specializes in parallel task execution, enabling fast processing and analysis of big data tasks such as video processing [18, 30]. Its in-memory computing capability reduces disk access, significantly boosting processing speeds compared to alternatives such as the Hadoop Distributed File System [31]. Spark supports multiple programming languages and offers various libraries for data processing tasks, such as PySpark.

*2.1.1 Partitioning .* A key component of Apache Spark concerns partitioning which is fundamental in determining the processing speed of the job at hand. Partitioning involves splitting up the data and tasks into smaller partitions that will ultimately be processed in parallel across the various nodes [22]. Ensuring a balanced distribution of tasks across partitions is critical to prevent any single node from becoming a bottleneck due to an excessive workload compared to others.

In this paper, the frames will be split up into different partitions that will be distributed across the various nodes. Traditional approaches typically divide data into partitions of equal size, basing the division on the quantity of data [8], which can be useful for tabular or time series data. However, in this research, information from each data point (i.e., the video frames) will be leveraged to dynamically create the partitions and better split up the workload.

## 2.2 Dataset

In this paper, the MIT DriveSeg dataset, which contains images of different driving scenarios that were acquired using a camera mounted on top of a car during the daylight was utilized. The dataset contains 20,100 video frames from 68 different driving scenarios [6]. The different scenarios depict different situations such as driving through a busy downtown street with a large number of potential objects or driving on a rural road with very few objects. The images have annotations for the different objects that have been labeled using semi-automated methods [7]. Some annotations may not be accurate because of semi-automatic labeling. Therefore, the annotations and labels have not been used in this paper.

## 2.3 Object Detection Techniques

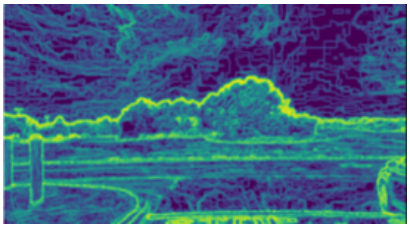
As mentioned in Section 2.2, the annotations that came with the MIT DriveSeg dataset, were shown not to be accurate due to the use of semi-automated methods which provided the segmentation of the images. Therefore, in this work, object detection of the driving scenarios is done through the utilization of bounding boxes. A bounding box is a rectangular frame that can be used to identify objects within a given video frame [17].

*2.3.1 YOLO Pretrained Model.* The YOLOv3 model was used to eliminate the need to train a deep learning model from scratch. The use of YOLOv3 resulted in saving time by eliminating the need for precise labels and allowing for the implementation of transfer learning.

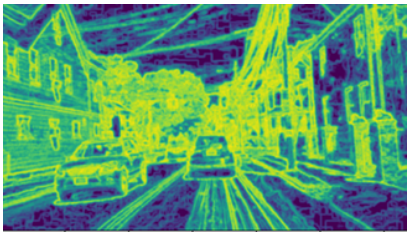
YOLO models are typically trained and optimized on a large dataset such as the Common Objects in Context (COCO) dataset [16]. Then, the model's weights can be further optimized or reused in other smaller datasets for object detection. YOLOv3 has shown improved metrics such as mean Average Precision (mAP) and also better processing speed when compared with other YOLO and pretrained models [16]. Additionally, YOLOv3 has been effective in maintaining a balance between detection speed and accuracy [13]. The YOLOv3 model has 80 possible classes of objects that can be detected within each image [12]. YOLOv3 is capable of detecting various objects that can be encountered in a driving scenario, such as cars, traffic lights, buses, trucks, street signs, pedestrians, and more. Each object found in the image can be represented using a bounding box.

*2.3.2 Entropy .* As mentioned in Section 2.1.1, one of the objectives of this research is to dynamically create partitions based on the estimated workload. A frame that has an abundant number of objects will take longer to process compared to a frame that has fewer objects. Therefore, there is a need to swiftly estimate the workload so that it can be split up into balanced partitions.

$$Entropy = -\sum_{i=1}^{L} p_i \log_2(p_i) \tag{1}$$



**(a) Frame with low entropy**



**(b) Frame with high entropy**

**Figure 1: Two different driving scenarios illustrating low and high entropy.**

Entropy can be used to determine the uncertainty or complexity in an image. Entropy can also be mathematically defined as the probability of occurrence $p_i$ at gray level $i$ where $L$ is the maximum pixel value [5, 19, 26]. Overall, equation 1 is computing the uncertainty in each pixel of the frame. Furthermore, local entropy can be used for images to examine the variance in images given a window or a neighbourhood [26]. In Figure 1, there are two frames from the MIT DriveSeg shown with their entropy overlaid on top of each other. Pixels that are brighter have a higher local entropy and vice versa. Homogeneous and less complex images (see Figure 1a) have a lower entropy when compared to heterogeneous and more complex images (see Figure 1b). Therefore, entropy can serve as a quick indicator of the number of objects in a frame, under the assumption that more complex images typically contain a greater number of objects, and simpler ones contain fewer.

## 3 PROPOSED APPROACH

This section will provide the overall approach used in devising the proposed technique and a description of how the proof-of-concept (POC) was implemented.

### 3.1 System Design

This system POC for the proposed technique was hosted on Amazon Web Services (AWS). AWS is a cloud platform which provides scalable computing and storage services [3]. AWS offers an ideal platform for rapidly deploying different resources which can have varying hardware specifications like the number of cores.

The MIT DriveSeg dataset was first uploaded to a Simple Storage Service (S3) bucket which is a durable and scalable data storage tool in AWS [4]. After the data was uploaded, a Spark cluster was created. AWS provides a service known as Elastic Map Reduce (EMR) which can be used to create and manage a Spark cluster with a main node and multiple worker nodes. EMR uses the Hadoop Yet Another Resource Negotiator (YARN) manager to organize the different nodes. The main node requests different resources such as memory or CPU cores from the YARN manager which then distributes the tasks among the worker nodes [4].

Each node in the cluster represents an AWS Elastic Cloud Computing (EC2) instance which is a cloud server containing various libraries including Python and Spark [4]. Additionally, PySpark, the Python library for Apache Spark, is installed on every node in the cluster, serving as the main library for the program in this paper. EC2 instances can be scaled up to include varying number of nodes which have varying numbers of cores. For the POC, the m5 family of EC2 instances which can have anywhere from 2 to 96 cores and from 8 to 384 GB of memory for each node were used [2].

Figure 2 gives an illustration of the system and can be described with the following steps.

(1) The EMR cluster will read from the S3 bucket containing frames from a driving scenario.
(2) The main node will load in the YOLOv3 model with weights trained on the COCO dataset using TensorFlow, a Python deep learning library which can be used to create and load neural network models [14].
(3) The YOLOv3 model is broadcast in the main node. In Spark, memory-intensive variables can be broadcast to provide a read-only version on the main node instead of replicating them across all worker nodes [29].
(4) The tasks and frames are submitted to the main node. The frames are split into partitions based on the chosen partitioning algorithm that are outlined in Table 1.
(5) The YARN manager performs task scheduling with the worker nodes.
(6) The worker nodes concurrently detect the objects in their assigned frames using the YOLOv3 model.
(7) The bounding box detections are sent back to the main node.
(8) Once all worker nodes are done with their tasks, the results are stored in a S3 bucket.

### 3.2 Experimental Design

The experiments used the one-factor-at-a-time approach where all the factors in the experiment are set to their default values while one factor was varied at a time [27]. The processing time required by the worker nodes to complete all of the detections is the measured metric for evaluating performance. The measurement of processing time is the duration from the moment the frames are dispatched to the worker nodes until the completion of all detections. The one-factor-at-a-time approach helps in identifying the key factors that impact the processing time. The different factors and their values are shown in Table 1, with the default value shown in bold. The selection of each factor will be further explained in this section.

*3.2.1 Distribution Framework.* The distribution framework is a simple factor that will compare whether the use of distributed computing is useful for the problem. Using no distribution will mean running the program on a single node and with no parallelism which will be compared to running it using multiple nodes on Apache Spark.

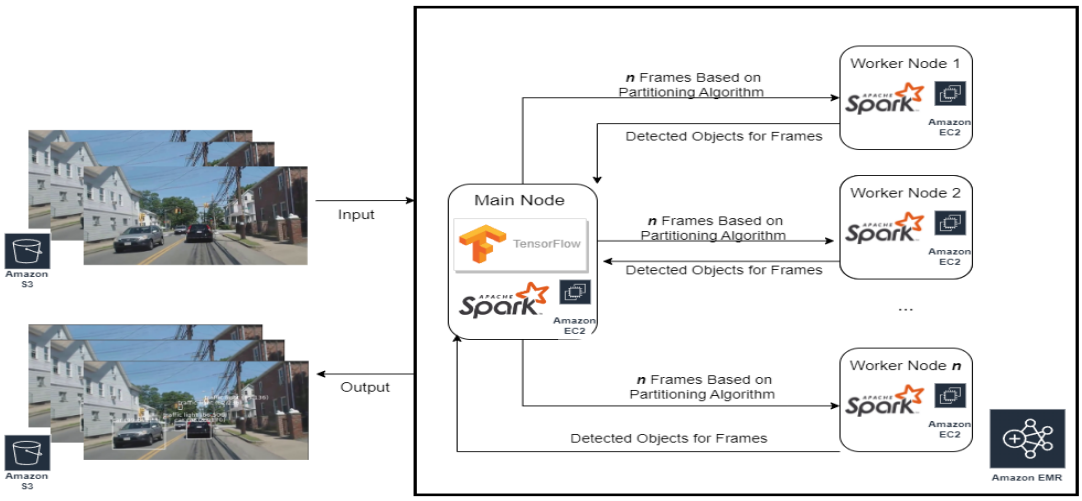



**Figure 2: Overall system design for object detection.**

*3.2.2 Number of Objects .* The number of objects within the scenario will be compared to evaluate whether this led to a difference in processing time. The hypothesis that needs to be verified is that as the average number of objects in a scenario increases, it would increase the overall workload for the system. For example, a driving scenario in a busy urban street, would require a lengthier duration for object detections, unlike a sparsely populated rural road with few objects.

*3.2.3 Number of Worker Nodes .* The number of worker nodes will be varied to analyze the impact of parallelism on performance.

*3.2.4 Number of CPU Cores.* Similar to Section 3.2.3, the number of CPU cores within each worker node will be changed to investigate the impact of core parallelism on performance.

*3.2.5 Total Number of Frames .* The total number of frames in a video being processed will be varied to assess concomitant changes in throughput. System throughput is defined as the number of frames processed per second [27].

*3.2.6 Partitioning Algorithm .* As mentioned in Section 2.1.1, one of the important objectives of this paper is to experiment with different partitioning algorithms. The number of partitions in Spark can be set manually. In most cases, setting the number of partitions to the number of total cores has been observed to yield good results [9].

For experimenting with this parameter, two different video scenarios were combined; one scenario concerning a busy street with a lot of objects and the other concerning a rural road with fewer objects. The combined video is meant to replicate a scenario where the number of objects varies throughout the video. For this case, partition splits should be based on the estimated workload in each partition and not based on the number of frames. One partition could have frames from the scenario that has numerous objects meaning it could become the performance bottleneck for the overall system. This is because as mentioned in Section 3.2.2, as the number of objects increase in a frame, the overall processing time for the frame is expected to increase.

The four partitioning algorithms are experimented with include the following.

(i) **Video-based**: In Video-based partitioning, each partition split will have frames from the same video scenario.
(ii) **Spark-based**: In Spark-based partitioning, the partitions will be assigned frames using the default Spark settings. Spark's default partitioning mechanism uses Hash Partitioning [9]. For this, a hash function is used to randomly assign the frame to a partition. However due to its randomness, it could result in a skewed data distribution as some partitions may contain larger portions of the data while some partitions could be empty [25, 28].
(iii) **Object-based**: In Object-based partitioning, each partition will have an equal number of frames from each video. Therefore, this implies that each partition should contain roughly an equal number of objects, based on the assumption that every frame from the each scenario possesses a similar quantity of objects. To estimate the workload, one random frame was sampled from each scenario with the assumption that frames with higher number of objects will have a higher workload and vice versa. To minimize overheads, the YOLOv3 model was deployed on this one frame to estimate whether there is a high or low number of objects within the whole scenario. A threshold of 7 objects was established through experimentation, categorizing a frame with 7 or more objects as an indicator of high workload and and as an indicator of low workload otherwise.
(iv) **Entropy-based**: As mentioned in Section 2.3.2, entropy can be used to estimate the workload resulting from a given frame, using the observation that frames with more objects will generally have a larger entropy value than those that do not. Therefore, similar, to Object-based partitioning, Entropy-based partitioning can be used to estimate the workload based on the contents of the frames. One of the benefits of using Entropy-based partitioning is its fast computation

speed, meaning it can be applied for all the frames. In contrast, Object-based partitioning can only be applied to one frame since it requires running the YOLOv3 model to get an estimate of the number of objects, which requires significantly more computation power compared to calculating the entropy of a frame. Additionally, to minimize system overheads, it is necessary to select a simple metric like entropy for estimating workload. Entropy-based partitioning is presented in *Algorithm 1*. First, in *Algorithm 1*, the videos and their frames are loaded into a variable known as *df* as shown in lines 1 to 4. Then the entropy of each frame is found on line 5. From lines 6 to 12, the frames are classified as having a low or high number of objects based on if they have a low or high entropy. The threshold value, found experimentally, will distinguish whether a frame has more or less objects. Once the groups are made, each partition will contain an equal number of *LESS_OBJECTS* and *MORE_OBJECTS* frames, as shown in lines 13 and 14.

**Algorithm 1** High Level Algorithm for Partitioning Frames Based on Entropy

```
1:  for video in videos do
2:      files = files.append(load_video(video))
3:  end for
4:  df = spark.read.format("binaryFile").load(files)
5:  df["entropy"] = getEntropy(df["frames"])
6:  for row in df do
7:      if row["entropy"] < THRESHOLD then
8:          row["group"] = LESS_OBJECTS
9:      else
10:         row["group"] = MORE_OBJECTS
11:     end if
12: end for
13: partitions = df.rdd.getNumPartitions()
14: df = partitionFrames(df, partitions)
```

## 4 EXPERIMENTAL RESULTS

The results for the experiments will be summarized. Each experiment was executed 10 times to calculate a mean and a standard deviation for the processing time.

**Table 1: System and Workload Parameters.**

| Factor | Value |
|---|---|
| Distribution Framework | (None, **Apache Spark**) |
| Number of Objects | (1-2, **7-8**, 12-13) |
| Number of Worker Nodes | (1, 2, **3**, 4) |
| Number of CPU Cores in Worker Nodes | (4, **8**, 16, 32) |
| Number of Total Frames | (150, **300**, 900, 2700) |
| Partitioning Algorithm | (Video, **Spark**, Object, Entropy |

Figure 3 shows that using Spark results in 50% decrease in processing time, compared with using no distribution framework. This establishes that using distributed computing is proven to be beneficial for the given problem.

Figure 4 depicts that as the number of objects increase in the frames, the processing time increases. This means that the workload in the system is related to the average number of objects in each scenario. This observation is important as it forms a necessary criterion for differentiating between various partitioning algorithms.

Figure 5 show that as the computation power increases, the processing times decreases. As the number of worker nodes increases, the performance improves. Similar results were found when increasing the number of cores.

Figure 6 shows that as the number of frames increase, the system throughput also increases. This is an expected result because as the size of the dataset increases, the throughput is expected to increase as the impact on overall performance of fixed overheads due to Spark, decreases with a larger dataset [1, 15].

Figure 7 shows the comparative performance of the different partitioning algorithms for two different scenarios, one on a busy street with a large number of objects and one on a rural street with no objects. By combining these scenarios a simulated dynamic video is created where the number of objects are varying from one frame to another. As shown, the worst performance was achieved by the Video-based algorithm where each partition contains frames from only one scenario which leads to uneven workload distribution. In Video-based partitioning, certain partitions may turn into bottlenecks due to the allocation of frames from scenarios with higher workloads. Spark's default partitioning algorithm was the second worst in terms of performance. As described in Section 3.2.6, Spark's default partitioning algorithm relies on a random Hash partitioner which may lead to a skewed data distribution. The randomness in results can be seen as the standard deviation after 10 runs of the algorithm is high, compared to the other algorithms analyzed in this research. This implies that relying on Spark's default algorithm for data distribution in a dynamic video scenario may not always give effective results. Object-based partitioning produced the best results in Figure 7. However, as mentioned in Section 3.2.6, using a single frame from each video to assess workload may lead to inaccuracies in dynamic scenarios and may not be a good choice to use for workload partitioning. If the video has varying number of objects, then it is not sufficient to use one frame to estimate the workload. Entropy-based partitioning produced comparable performance results as Object-based. The entropy for each frame was computed, making this approach the most accurate partitioning algorithm as it is frame-specific. Furthermore, as shown in Figure 7, Entropy-based partitioning achieved an improvement of ~13% in processing time when compared to Spark's default partitioning algorithm.

As mentioned in Section 3.2.6, for Entropy-based partitioning a threshold parameter needs to be set so that the workload of the frame can be classified. To experimentally find the threshold value, three scenarios that had a large number of objects and three scenarios that had a few objects were analyzed to calculate the entropy of each frame (see Figure 8). Each bin in the histogram is the number of frames that correspond to the energy value captured in the x-axis. As shown in the figure, there is a clear difference

between the two groupings. From these results, the threshold value used in *Algorithm 1* was set at $2.5 \times 10^6$, since this is the value that can differentiate a high workload frame from a low workload frame.

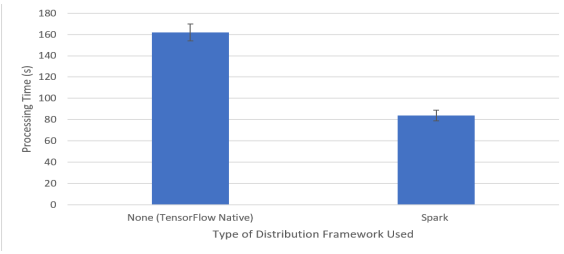



**Figure 3: Processing time for object detection versus distribution framework.**

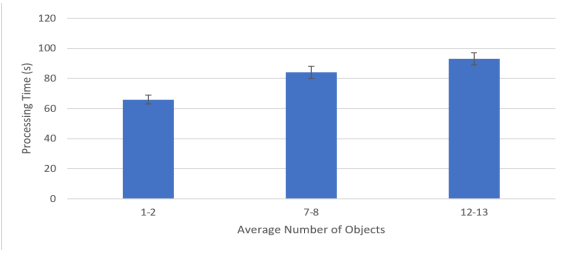



**Figure 4: Processing time for object detection versus average number of objects in each frame.**

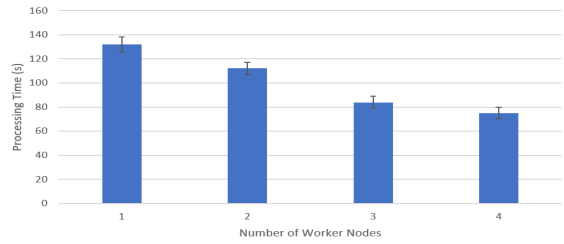



**Figure 5: Processing time for object detection versus number of worker nodes.**

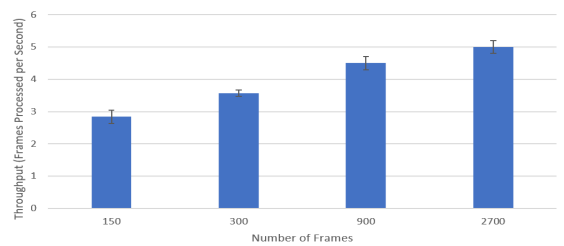



**Figure 6: Throughput versus total number of frames being processed.**

## 5 CONCLUSIONS

This paper demonstrated how parallel processing techniques deployed on Apache Spark can be used to improve performance of the deep learning based YOLOv3 model for detecting objects in videos. First it was shown that using parallel processing is significantly faster than using only TensorFlow with no parallel programming. The results showed that as the number of objects in the videos increased, the processing time also increased, this demonstrating the correlation between the workload intensity and the number of objects in the scenarios. As the computation power is increased by adding more worker nodes or CPU cores, the performance improved as well. Throughput was also found to be related to the size of the videos. Different partitioning algorithms for dynamic workload distribution were studied. The Entropy-based partitioning algorithm showed improved performance in processing the detections when compared to Spark's random partitioning.

## 6 FUTURE WORK

The preliminary results presented in this research open up several items for future work which will include the following:

(i) The investigation of longer videos is one of the important components of future work. For the initial work presented in this pilot project, each video was 300 frames long. In the results, it was shown that as the size of the dataset increases, parallel processing becomes more beneficial. Therefore, experimentation with larger videos, that give rise to larger datasets, need to be carried out. Furthermore, longer videos

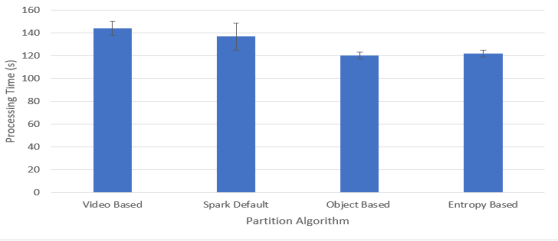



**Figure 7: Processing time for detection for different partitioning algorithm.**

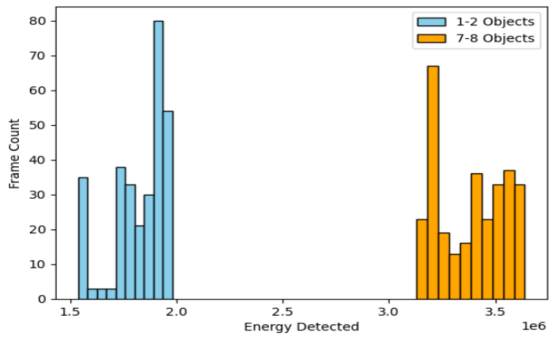



**Figure 8: Distribution of entropy grouped by number of objects.**

that have dynamically changing scenes need to be used to further confirm the usefulness of Entropy-based partitioning.

(ii) The removal of redundant frames is expected to improve system performance. In this paper, every single frame in the video was used for detection; however, many of these frames have overlapping information. Performance would improve by sampling a fixed number of frames once a substantial change has taken place. Algorithms for identifying redundant frames will be investigated. Improving the latency of object detection will also be useful for real time applications.

(iii) Incorporating the use of other deep learning models will be another direction for future work. In this paper, the YOLOv3 model was used for object detection but there are newer versions of YOLO models, the performance of which can be compared with the results from this research. Other pre-trained models such as EfficientNet will also be investigated.

(iv) Computing the accuracy of object detections would be included in future work. In this research, the accuracy of the bounding boxes were not computed. Metrics like mean Average Precision (mAP) and Jaccard index are used to determine the accuracy of the bounding box. Analysis of the potential trade-offs between speed and accuracy will be performed.

(v) Future work will also look to distribute pixels from individual frames among the different nodes using parallel processing.

(vi) The use of Graphical Processing Units (GPUs) will be used in future work to study its impact in performance, given their proven ability to speed up processing times in deep learning applications [27].

## ACKNOWLEDGMENTS

This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

## REFERENCES

[1] Nasim Ahmed, Andre LC Barczak, Teo Susnjak, and Mohammed A Rashid. 2020. A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench. *Journal of Big Data* 7, 1 (2020), 1–18.

[2] AWS. [n. d.]. Amazon EC2 M5 Instances. https://aws.amazon.com/ec2/instance-types/m5/

[3] Ignacio Bermudez, Stefano Traverso, Marco Mellia, and Maurizio Munafo. 2013. Exploring the cloud from passive measurements: The Amazon AWS case. In *2013 Proceedings IEEE INFOCOM*. IEEE, 230–234.

[4] Lin Chen, Rui Li, Yige Liu, Ruixuan Zhang, and Diane Myung-kyung Woodbridge. 2017. Machine learning-based product recommendation using Apache Spark. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 1–6.

[5] Sandipan Dey. 2018. *Hands-On Image Processing with Python: Expert techniques for advanced image analysis and effective interpretation of image data.* Packt Publishing Ltd.

[6] Li Ding, Michael Glazer, Jack Terwilliger, Bryan Reimer, and Lex Fridman. 2020. MIT DriveSeg (Semi-auto) Dataset. https://doi.org/10.21227/nb3n-kk46

[7] Li Ding, Jack Terwilliger, Rini Sherony, Bryan Reimer, and Lex Fridman. 2020. MIT DriveSeg (Semi-auto) Dataset: Large-scale Semi-automated Annotation of Semantic Driving Scenes. *Massachusetts Institute of Technology AgeLab Technical Report* 2 (2020).

[8] J Geetha and N. G. Harshit. 2019. Implementation and Performance Comparison of Partitioning Techniques in Apache Spark. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 1–5. https://doi.org/10.1109/ICCCNT45670.2019.8944759

[9] Anastasios Gounaris, Georgia Kougka, Ruben Tous, Carlos Tripiana Montes, and Jordi Torres. 2017. Dynamic Configuration of Partitioning in Spark Applications. *IEEE Transactions on Parallel and Distributed Systems* 28, 7 (2017), 1891–1904. https://doi.org/10.1109/TPDS.2017.2647939

[10] Edward Helmore. 2022. Tesla behind eight-vehicle crash was in "full self-driving" mode, says driver. https://www.theguardian.com/technology/2022/dec/22/tesla-crash-full-self-driving-mode-san-francisco

[11] Bowen Li, Nat Shineman, Jayson Boubin, and Christopher Stewart. 2021. Comparison of Object Detectors for Fully Autonomous Aerial Systems Performance. In *Companion of the ACM/SPEC International Conference on Performance Engineering* (Virtual Event, France) *(ICPE '21)*. Association for Computing Machinery, New York, NY, USA, 165–166. https://doi.org/10.1145/3447545.3451170

[12] Tao Li, Yitao Ma, and Tetsuo Endoh. 2020. A Systematic Study of Tiny YOLO3 Inference: Toward Compact Brainware Processor With Less Memory and Logic Gate. *IEEE Access* 8 (2020), 142931–142955. https://doi.org/10.1109/ACCESS.2020.3013934

[13] Ignacio Martinez-Alpiste, Gelayol Golcarenarenji, Qi Wang, and Jose Maria Alcaraz-Calero. 2021. A dynamic discarding technique to increase speed and preserve accuracy for YOLOv3. *Neural Computing and Applications* 33, 16 (2021), 9961–9973.

[14] Bo Pang, Erik Nijkamp, and Ying Nian Wu. 2020. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics* 45, 2 (2020), 227–248.

[15] Md Armanur Rahman, J Hossen, and C Venkataseshaiah. 2018. SMBSP: a self-tuning approach using machine learning to improve performance of spark in big data processing. In *2018 7th International Conference on Computer and Communication Engineering (ICCCE)*. IEEE, 274–279.

[16] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. arXiv:1804.02767 [cs.CV]

[17] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. 2019. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 658–666.

[18] Sajad Sameti, Mea Wang, and Diwakar Krishnamurthy. 2018. Stride: Distributed video transcoding in spark. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.

[19] scikit image. [n. d.]. Entropy. https://scikit-image.org/docs/stable/auto_examples/filters/plot_entropy.html

[20] Arindrajit Seal and Arindam Mukherjee. 2019. Real Time Accident Prediction and Related Congestion Control Using Spark Streaming in an AWS EMR cluster. In *2019 SoutheastCon*. 1–7. https://doi.org/10.1109/SoutheastCon42311.2019.9020661

[21] David Shepardson. 2023. GM's cruise recalling 950 driverless cars after pedestrian dragged in ... https://www.reuters.com/business/autos-transportation/gms-cruise-recall-950-driverless-cars-after-accident-involving-pedestrian-2023-11-08/

[22] Tinku Singh, Shivam Gupta, Manish Kumar, et al. 2023. Performance analysis and deployment of partitioning strategies in apache spark. *Procedia Computer Science* 218 (2023), 594–603.

[23] Lauren Smiley. 2023. The legal saga of Uber's fatal self-driving car crash is over. https://www.wired.com/story/ubers-fatal-self-driving-car-crash-saga-over-operator-avoids-prison/

[24] Hayley Smith and Russ Mitchell. 2022. A Tesla on autopilot killed two people in Gardena. is the driver guilty of manslaughter? https://www.latimes.com/california/story/2022-01-19/a-tesla-on-autopilot-killed-two-people-in-gardena-is-the-driver-guilty-of-manslaughter

[25] H. S. Sreeyuktha and J. Geetha Reddy. 2019. Partitioning in Apache Spark. In *Innovations in Computer Science and Engineering*, H. S. Saini, Rishi Sayal, Aliseri Govardhan, and Rajkumar Buyya (Eds.). Springer Singapore, Singapore, 493–498.

[26] Badri Narayan Subudhi, Pradipta Kumar Nanda, and Ashish Ghosh. 2011. Entropy based region selection for moving object detection. *Pattern recognition letters* 32, 15 (2011), 2097–2108.

[27] Azhar Talha Syed and Shikharesh Majumdar. 2022. Parallel Processing Techniques for Analyzing Large Video Files: a Deep Learning Based Approach. In *2022 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCloud/SocialCom/SustainCom)*. 270–279. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom57177.2022.00041

[28] Zhuo Tang, Wei Lv, Kenli Li, and Keqin Li. 2018. An intermediate data partition algorithm for skew mitigation in spark computing environment. *IEEE Transactions on Cloud Computing* 9, 2 (2018), 461–474.

[29] Isaac Triguero, Mikel Galar, D Merino, Jesus Maillo, Humberto Bustince, and Francisco Herrera. 2016. Evolutionary undersampling for extremely imbalanced big data classification under apache spark. In *2016 IEEE congress on evolutionary computation (CEC)*. IEEE, 640–647.

[30] Md Azher Uddin, Aftab Alam, Nguyen Anh Tu, Md Siyamul Islam, and Young-Koo Lee. 2019. SIAT: A distributed video analytics framework for intelligent video surveillance. *Symmetry* 11, 7 (2019), 911.

[31] Ankush Verma, Ashik Hussain Mansuri, and Neelesh Jain. 2016. Big data management processing with Hadoop MapReduce and spark technology: A comparison. In *2016 symposium on colossal data analysis and networking (CDAN)*. IEEE, 1–4.

[32] Haijun Zhang, Mingshan Sun, Qun Li, Linlin Liu, Ming Liu, and Yuzhu Ji. 2021. An empirical study of multi-scale object detection in high resolution UAV images. *Neurocomputing* 421 (2021), 173–182.

# Evaluating Emerging AI/ML Accelerators: IPU, RDU, and NVIDIA/AMD GPUs

Hongwu Peng*
University of Connecticut
Storrs, CT, USA
hongwu.peng@uconn.edu

Caiwen Ding
University of Connecticut
Storrs, CT, USA
caiwen.ding@uconn.edu

Tong Geng
University of Rochester
Rochester, NY, USA
tgeng@ur.rochester.edu

Sutanay Choudhury
Pacific Northwest National Laboratory
Richland, WA, USA
sutanay.choudhury@pnnl.gov

Kevin Barker
Pacific Northwest National Laboratory
Richland, WA, USA
kevin.barker@pnnl.gov

Ang Li
Pacific Northwest National Laboratory
Richland, WA, USA
ang.li@pnnl.gov

## ABSTRACT

The relentless advancement of artificial intelligence (AI) and machine learning (ML) applications necessitates the development of specialized hardware accelerators capable of handling the increasing complexity and computational demands. Traditional computing architectures, based on the von Neumann model, are being outstripped by the requirements of contemporary AI/ML algorithms, leading to a surge in the creation of accelerators like the Graphcore Intelligence Processing Unit (IPU), Sambanova Reconfigurable Dataflow Unit (RDU), and enhanced GPU platforms. These hardware accelerators are characterized by their innovative data-flow architectures and other design optimizations that promise to deliver superior performance and energy efficiency for AI/ML tasks.

This research provides a preliminary evaluation and comparison of these commercial AI/ML accelerators, delving into their hardware and software design features to discern their strengths and unique capabilities. By conducting a series of benchmark evaluations on common DNN operators and other AI/ML workloads, we aim to illuminate the advantages of data-flow architectures over conventional processor designs and offer insights into the performance trade-offs of each platform. The findings from our study will serve as a valuable reference for the design and performance expectations of research prototypes, thereby facilitating the development of next-generation hardware accelerators tailored for the ever-evolving landscape of AI/ML applications. Through this analysis, we aspire to contribute to the broader understanding of current accelerator technologies and to provide guidance for future innovations in the field.

*Work done during an internship at Pacific Northwest National Laboratory.

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only. Request permissions from owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651428

## CCS CONCEPTS

• **Computer systems organization** → *Architectures*; • **Computing methodologies** → **Parallel computing methodologies**; **Machine learning**; **Parallel computing methodologies**; **Machine learning**;

## KEYWORDS

High-Performance Computing, Dataflow architecture, Performance benchmarking

**ACM Reference Format:**
Hongwu Peng*, Caiwen Ding, Tong Geng, Sutanay Choudhury, Kevin Barker, and Ang Li. 2024. Evaluating Emerging AI/ML Accelerators: IPU, RDU, and NVIDIA/AMD GPUs . In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651428

## 1 INTRODUCTION

The rapid expansion of artificial intelligence (AI) and machine learning (ML) applications has led to a paradigm shift in computational hardware design. Traditional von-Neumann architectures, which have served as the backbone of computing for decades, are increasingly challenged by the demands of modern AI/ML workloads. These workloads often involve complex operations such as matrix multiplications, convolutions, and graph processing, which can be highly parallelizable but are bottlenecked by the data transfer constraints inherent in the von-Neumann architecture. To address these challenges, there has been a surge in the development of specialized hardware accelerators that aim to optimize the performance of AI/ML tasks through innovative architectural designs and execution models.

Among the emerging contenders in the field of AI/ML accelerators, Graphcore's Intelligence Processing Unit (IPU) [8] and Sambanova's Reconfigurable Dataflow Unit (RDU) [25] stand out for their unique approach to hardware acceleration. These platforms leverage data-flow architectures, which are fundamentally different from the von-Neumann architecture, to enable more efficient computation for AI/ML workloads. By aligning the hardware design with the data-centric nature of AI/ML algorithms, these accelerators promise significant gains in performance and energy efficiency.

**Table 1: Device information of Graphcore IPU, Sambanova RDU, and Nvidia/AMD GPU**

| Device | TSMC Process | Die size (mm^2) | Transistors | Archi. | On-chip SRAM (MB) | Off-chip Memory | Clk (GHz) | FP64 (TFLOPs) | FP32 (TFLOPs) | FP16 (TFLOPs) | Power |
|---|---|---|---|---|---|---|---|---|---|---|---|
| IPU GC2000 | 7nm | 823 | 59.4 B | IPU (MIMD) | 900@Scratchpad | 448 GB@DRAM 20 GB/s | 1.33 | \ | 62.5 (AMP) | 250 (AMP) | 165 W |
| SN10 RDU | 7nm | N/A | 40 B | RDU (CGRA) | 320@PMU | 1.5 TB@DRAM 100 GB/s | N/A | \ | \ | 325 | N/A |
| Nvidia V100 | 12nm | 815 | 21.1 B | Volta (SIMT) | 10.2@L1 Cache 6.1@L2 Cache | 16 GB@HBM2 1.13 TB/s | 1.41 | 7.8 (CUDA Core) | 15.7 (CUDA Core) | 125 (Tensor Core) | 250 W |
| Nvidia A100 | 7nm | 826 | 54.2 B | Ampere (SIMT) | 24.6@L1 Cache 40.9@L2 Cache | 40GB@HBM2 1.6 TB/s | 1.6 | 9.7 (CUDA Core) 19.5 (Tensor Core) | 19.5 (CUDA Core) | 312 (Tensor Core) | 250 W |
| AMD MI100 | 7nm | 750 | 25.6 B | CDNA (SIMT) | 1.92@L1 Cache 8@L2 Cache | 32 GB@HBM2 1.23 TB/s | 1.5 | 11.5 (CU Core) | 23.1 (CU Core) 46.14 (Matrix core) | 184.57 (Matrix Core) | 290 W |

In addition to these specialized data-flow accelerators, Graphics Processing Units (GPUs) have also been at the forefront of AI/ML acceleration. With their highly parallel structure and robust ecosystem, GPUs continue to evolve with features such as Tensor Cores and enhanced memory hierarchies to better support the intensive computational demands of AI/ML applications.

In this study, we aim to provide a comprehensive evaluation and comparison of these commercial AI/ML accelerators. Our research delves into the architectural intricacies of the Graphcore IPU, Sambanova RDU, and various GPU platforms, examining their system design, memory hierarchy, computing resources, and programming models. By conducting a series of benchmark evaluations across a range of DNN operators, we seek to uncover the strengths and limitations of each platform, offering insights into their lower-precision floating-point numerical performance characteristics and suitability for different AI/ML tasks.

Our findings will serve as a valuable reference for both the academic and industrial communities, guiding the development of future hardware accelerators. By understanding the common strategies employed by current accelerators and identifying the unique features that contribute to their performance, we can inform the design of next-generation AI/ML hardware that is even more tailored to the requirements of emerging workloads. In doing so, we contribute to the ongoing quest for hardware architectures that can keep pace with the relentless advancement of AI/ML technologies.

## 2 EMERGING AI/ML ACCELERATORS

### 2.1 Graphcore IPU

**System Information.** The Graphcore Intelligence Processing Unit (IPU) system used in the test is the IPU-POD16 with four IPU-M2000 units [9]. The IPU-POD16 utilizes a Dell R6525 Poplar server with dual-socket AMD Epyc2 CPUs as the host server. Each IPU-M2000 unit (1U) comprises four GC200 IPU chips connected through IPU-Link with a bandwidth of 192 GB/s. The GC200 IPU chip contains 1472 independent IPU-tiles and can process up to 8832 separate program threads in parallel.

**Memory Resource and Hierarchy.** The architecture of the GC200 IPU chip is illustrated in Fig. 1(a). Each tile of the GC200 IPU chip [8] is equipped with 624 KiB of local scratchpad memory, and 1472 tiles contribute to a total of 900 MB of in-processor memory with 47.5 TB/s on-chip bandwidth for a single chip. Columns of IPU-tiles are connected through IPU-Exchange with 8 TB/s bandwidth [8], but with higher latency penalty [20]. Unlike cache, the scratchpad memory within an IPU-tile can be accessed irregularly without penalty. Each GC200 IPU chip features 10 IPU-Links and supports up to 320 GB/s chip-to-chip bandwidth. In addition to the abundant on-chip SRAM memory, each IPU-M2000 unit also offers 448 GB of streaming DDR memory with 20 GB/s bandwidth, which is used to store the dataset or output and support infrequent transactions.

**Computing Resource.** The GC200 IPU chip employs Accumulating Matrix Product (AMP) units for its floating-point computation. Each IPU-tile has one AMP unit and can perform up to 64 multiply-accumulate (MAC) operations per cycle. The theoretical computing throughput of a single GC200 IPU chip is 250 TFLOPs (with sparsity) for FP16 format and 62.5 TFLOPs (with sparsity) for FP32 format. A single IPU-M2000 unit can achieve up to 1 PetaFLOPs peak throughput for FP16 format.

**Architecture Information** IPUs offer a much larger core count than CPU platforms, and each IPU-tile is capable of executing completely distinct programs [20]. Compared to GPU platforms, IPU-tiles are connected with high-performance on-chip networks, providing greater flexibility for Multiple Instruction Multiple Data (MIMD) programming capability and better suitability for sparse and irregular processing. The IPU programming model follows the standard Bulk Synchronous Parallel (BSP) model [29]. The IPU execution flow consists of three steps: (1) local compute, (2) global synchronization, and (3) on-chip data exchange. During the compute phase, IPU-tiles operate on their local data. After the computation is completed, IPU-tiles synchronize before data exchange. The data exchange phase is supported by an on-chip interconnect with efficient point-to-point and collective communications [20]. Each IPU-tile has hardware support for six threads with Simultaneous Multi-Threading (SMT) [28] capability to use shared hardware resources in a round-robin fashion. SMT technology can effectively hide memory access and branch latency, increasing overall throughput.

**Compiler Stack & Parallelism Support.** The Graphcore IPU utilizes the Poplar compiler stack [11] for runtime deployment and optimization. For machine learning workloads, popular frameworks such as TensorFlow and PyTorch are used as front-ends for easier development. The Poplar Advanced Run Time (PopART) serves as the interface between the machine learning front-end and the Poplar Graph Compiler. The output of the Poplar Graph Compiler is further processed by the Graph Element Compiler to exploit efficient computation and computation patterns according to the

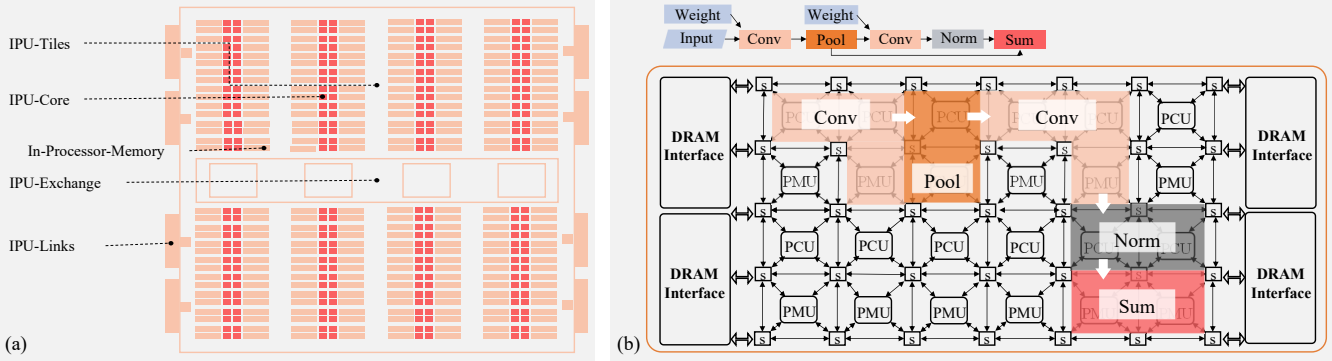



**Figure 1: (a) Graphcore IPU architecture. (b) Sambanova RDU architecture**

BSP model. IPUs have abundant instruction set support for various applications, and Graph Elements can be written using C/C++ with LLVM compiler or IPU assembly. The current version of IPU supports task-level parallelism [10] across multiple IPUs. However, only a limited form of task parallelism can be used within an IPU chip to overlap the IO latency. Consequently, single-chip IPU execution flow optimizes individual operator performance and has limited support for operator-wise parallelism.

## 2.2 Sambanova RDU

**System Information.** The Sambanova Reconfigurable Dataflow Unit (RDU) system SN10-8 used in the test is built upon eight Cardinal SN10 RDUs [23, 24]. The SN10 RDU adopts a coarse-grain reconfigurable array (CGRA) data-flow architecture [13] for its hardware. As shown in Fig. 1(b), the basic units of RDU [25] include pattern compute units (PCUs), pattern memory units (PMUs), and pipelined switches for the on-chip network. Each PCU is equipped with six SIMD stages, and each stage has 16 single instruction multiple data (SIMD) lanes, totaling 96 Functional Units (FUs). Each SN10 chip contains 640 PCUs and 640 PMUs, providing 320 MB of on-chip SRAM. The PCU for the SN10 chip supports only the FP16 format for arithmetic computations, and the theoretical throughput of a single SN10 chip is 325 TFLOPs.

**Memory Resource and Hierarchy.** Each PMU scratchpad of the SN10 has 512 KB of memory and 16 banks, with the number of banks matching the number of PCU lanes to provide vectorized data access. The PMU supports several memory access modes [25] to facilitate different applications: strided-banking mode for dense operators, FIFO mode for streaming data, line-buffer mode for sliding window accesses, duplication mode to support parallel reads, and N-buffering mode for coarse-grained pipelines. In addition to on-chip memory resources, the SN10 can be connected to up to 1.5 TB of external DRAM for streaming and random accesses. The streaming access of external DRAM supports up to 100 GB/s bandwidth, while random (sparse) access supports only 12 GB/s bandwidth.

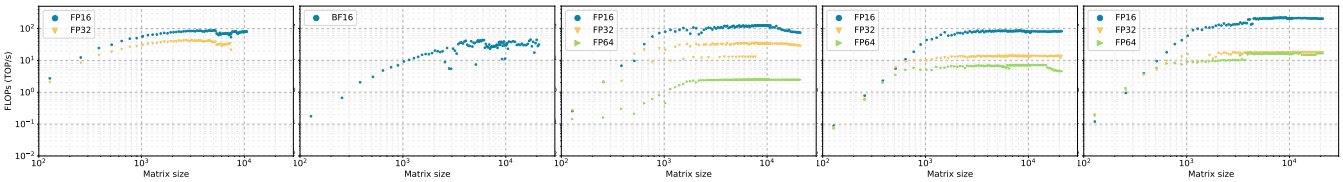
**Architecture Information.** The data-flow graph is mapped across PCU and PMU stages, utilizing vectorization to increase the parallelism level within each PCU lane. Each PCU has multiple SIMD lanes and stages, exploiting fine-grained parallelism for its computation tasks. The PCU consumes pipelined data from previous stages and generates pipelined data for later stages. Communication between PCUs and PMUs is based on streaming to avoid pipeline stalls and memory latency overhead. Reconfigurable controller blocks are distributed among PCUs and PMUs to match data stream rates and trigger instruction execution. Operators within the dataflow graph are mapped to multiple PCUs and PMUs depending on their size. The hardware mapping and scheduling of PCUs and PMUs to the data-flow graph aim to maximize throughput and minimize latency.

**Compiler Stack & Parallelism Support.** The Sambanova system utilizes SambaFlow [4] as the end-to-end compiler framework for machine learning acceleration. SambaFlow takes open-source frameworks such as PyTorch and TensorFlow as entry points to build the DNN model. The model is then fed into a dataflow graph analyzer to generate a dataflow graph with Spatial intermediate representation (IR) [21]. The dataflow graph analyzer conducts design space exploration and performs domain-specific optimizations, such as layer fusion for DNN model mapping on RDU hardware. If the operators are not present in existing ML frameworks, users can also specify new operators through the tensor index notation API. The template compiler maps the operator into an optimized dataflow implementation, called a spatial template, on RDU hardware. The final dataflow compiler and assembler layer performs final transformations such as parallelization, placement, and routing. The dataflow graph is then transformed into the final RDU hardware mapping, and the executable file is generated.

## 2.3 Nvidia & AMD GPU

**System Information.** We use Nvidia V100 [3] and A100 [2], as well as AMD MI100 and MI250 for the GPU platform benchmark evaluation. Detailed information such as L1 cache, L2 cache, global memory bandwidth, and theoretical throughput is provided in Tab. 1.

**Nvidia GPU Architecture** For Nvidia GPUs, the streaming multiprocessors (SM) serve as the basic unit for instruction execution and scheduling. An SM may have multiple SM partitions, which share the same global L1 data cache (shared memory) and L1 instruction cache. Each SM partition has its own register file, L0 instruction cache, warp scheduler, dispatch unit, and computing resources. Computing resources within the SM partition include




(a) Graphcore (b) Sambanova (c) MI100 (d) V100 (e) A100

**Figure 2: Cross platform evaluation on square GEMM operators.**

floating-point Compute Unified Device Architecture (CUDA) cores and Tensor Cores. Tensor Cores perform multiple FP16/FP32 mixed-precision fused multiply-add (FMA) operations within a single cycle and offer much higher computational throughput. Thirty-two threads are grouped into a warp and are mapped to a single SM partition for SIMD execution. Warp execution is overlapped through a fine-grained pipeline to hide instruction fetch or memory fetch latency. Multiple SM units are grouped into a single texture processing cluster (TPC), and multiple TPCs are grouped into a single GPU processing cluster (GPC). GPCs share a global L2 cache for on-chip data buffering.

**AMD GPU Architecture** AMD GPUs use the CDNA architecture [15, 16] for their MI100 and MI250 products. The architecture is similar to the Nvidia Volta/Ampere architecture. The CDNA architecture employs terms such as Compute Unit (CU), CU Core, and Matrix Core, as opposed to SMs, CUDA core, and tensor core for Nvidia GPUs. The CU uses the Graphics Core Next (GCN) architecture [17] and has multiple floating-point cores and a single Matrix Core with enhanced throughput.

**Memory Resource and Hierarchy.** To boost the data transfer rate between GPU on-chip resources and off-chip memory, High Bandwidth Memory (HBM) technology is widely adopted for both Nvidia and AMD GPUs. HBM consists of stacks of DRAM dies with through-silicon via (TSV) connections, providing up to several TB/s of external memory bandwidth. The connection of HBM dies with GPU dies is based on an underlying silicon interposer to ensure reliable and high-speed connections.

**Programming Model.** Nvidia employs the CUDA parallel programming model [1] for its GPUs to leverage their massive processing power with minimal implementation coding effort. It enables heterogeneous computation where the CPU and GPU have separate memory and thread spaces. The CUDA programming is based on kernel functions, which are called by the host CPU and executed on the GPU device. The kernel function is executed with massive concurrent threads on the GPU that share the same kernel code. The thread ID is used for memory addressing and thread cooperation. CUDA threads are extremely lightweight, allowing them to be created or switched with minimal penalties. Thirty-two threads are grouped into a single warp, and the warp scheduler determines the sequence of warp execution to hide instruction and memory access latency. To reduce global memory traffic, thread cooperation within a thread block is enabled through shared memory (L1 cache) and thread synchronization primitives. An L2 cache with residency control is also available to further reduce global memory access bandwidth and latency. Thread blocks to SMs mapping are scheduled during runtime, and multiple thread blocks can be mapped to a single SM. The CUDA programming model makes the CUDA code scalable for various hardware platforms, ranging from laptops to high-end servers with minimal changes.

AMD adopted ROCm [19] for its software stack and uses the Heterogeneous Interface for Portability (HIP) [18] programming model. HIP Runtime API and kernel programming methods can be used for both AMD and Nvidia GPU platforms. The HIP programming model is similar to the CUDA programming model.

**Compiler Support** Existing deep learning frameworks, such as TensorFlow and PyTorch, can be used with the CUDA and ROCm software stack to support automatic differentiation and DNN computational graph generation.

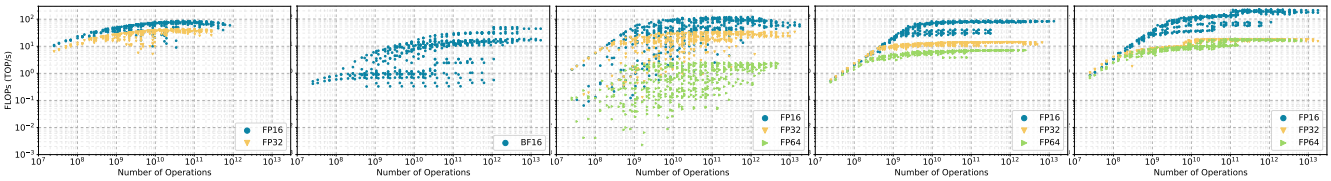
**Table 2: System information.**

| Device | SambaNov SN10 | GraphCore GC200 | NvidiaV100 |
|---|---|---|---|
| Host CPU | AMD EPYC 7742 | AMD EPYC 7302 | Intel E5-2620 |
| Software Stack | PyTorch 1.10.2<br>SambaFlow 1.14.0 | TensorFlow 1.15.5<br>Poplar 2.4.0 | PyTorch 1.12.1<br>CUDA 11.6 |
| Device | Nvidia A100 | AMD MI100 | AMD MI250 |
| Host CPU | AMD EPYC 7502 | AMD EPYC 7543 | |
| Software Stack | PyTorch 1.12.1<br>CUDA 11.6 | PyTorch 1.13.0<br>ROCm 5.3.0 | PyTorch 1.13.0<br>ROCm 5.3.0 |

## 2.4 Summary

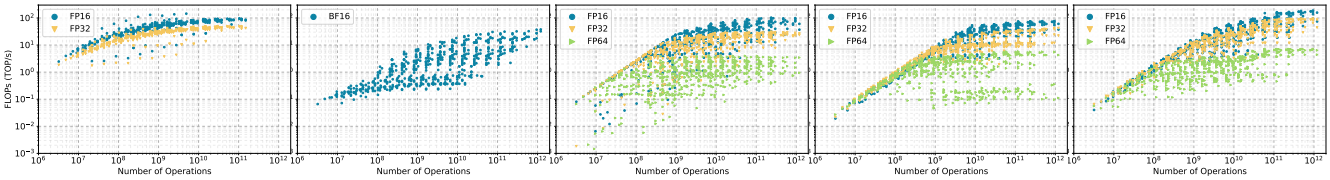
**Similarity Between Platforms** Graphcore IPU, Sambanova RDU, and GPU platforms each have their own basic SIMD/SIMT units: IPU-tile, PCU, and SM. These basic SIMD units share similar architecture and process lightweight concurrent threads while switching between thread groups (warps) with minimal penalty. Such SIMD architecture fully exploits fine-grained pipelined parallelism, enabling high throughput and high energy-efficient parallel computation.

**Difference Between Platforms** Although the three accelerator platforms' basic SIMD/SIMT units have similar architecture and execution models, the interconnect and scheduling between these units differ among platforms. Graphcore IPU features a global crossbar that connects every IPU-tile, and the scheduling and execution model between IPU-tiles follows the BSP model with three steps: local computation, synchronization, and data communication/exchange. Sambanova RDU provides reconfigurable switches



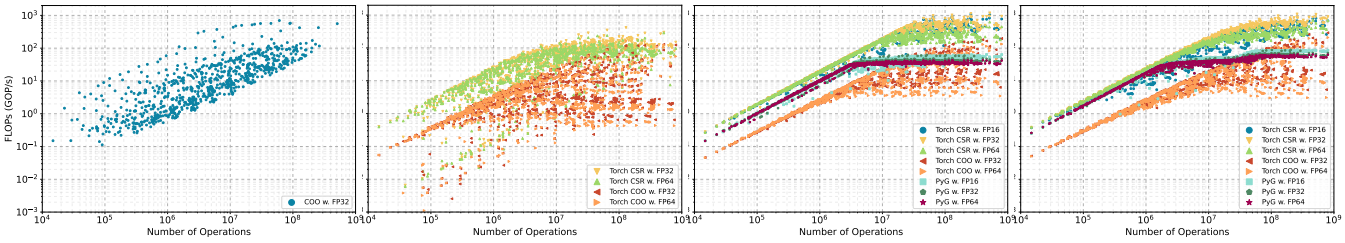

(a) Graphcore (b) Sambanova (c) MI100 (d) V100 (e) A100

**Figure 3: Cross platform evaluation on BERT operators.**




(a) Graphcore (b) Sambanova (c) MI100 (d) V100 (e) A100

**Figure 4: Cross platform evaluation on 2D convolution operators.**




(a) Graphcore (b) MI100 (c) V100 (d) A100

**Figure 5: Cross platform evaluation on SPMM operators.**

for PCU and PMU connections, where scheduling and mapping of PCUs and PMUs use the Spatial dataflow graph compiler for task parallelism. GPU architecture has limited connections and communication bridges between SMs, and the execution and mapping of tasks to the GPU follow the CUDA programming model with SIMT style. Overall, Graphcore IPU and Sambanova IPU platforms offer greater flexibility in SIMD/SIMT unit mapping and scheduling than the GPU platform, potentially providing more advantages for applications with sparse or irregular computation and communication patterns.

## 3 EVALUATION RESULTS

We conduct benchmark evaluations of several commonly used DNN operators on the target platforms, including ❶ general matrix multiplication (GEMM), ❷ 2D convolution (Conv2D), and ❸ sparse-matrix dense-matrix multiplication (SPMM). It is important to note that the Sambanova SN10 platform lacks compiler support for SPMM, and therefore is not evaluated for this operator.

### 3.1 Square GEMM Benchmark

We conduct benchmarking of square matrix multiplication following the software setup outlined in Table 2. The matrix size is varied from 256 to 10,624 with a 128 step size, and from 10,752 to 20,992 with a 512 step size. We record throughput performance over available floating-point formats for each platform. The Graphcore GC200 IPU platform is benchmarked for its available FP16 and FP32 data types, with matrix size limited to 10,496 due to limited on-chip memory. Both FP16, FP32, and FP64 throughputs were recorded for the GPU platforms. The evaluation results are shown in Fig. 2. It is worth noting that for the Sambanova platform, throughput

performance becomes unstable when matrix sizes grow to 2500 and above, due to unstable PCUs and PMUs mapping through the SambaFlow compiler. The AMD MI100 GPU platform shows unstable performance between matrix sizes 256 to 7936 for FP32 format due to choices of CU core and Matrix core during PyTorch to ROCm compilation.

**Table 3: Normalized geomean hardware throughput for GEMM.**

| Platform | A100 | V100 | MI100 | GC200 | SN10 |
|---|---|---|---|---|---|
| FP16 | 10.64 | 4.81 | 7.00 | 6.00 | 2.96 |
| FP32 | 1.30 | 1.00 | 2.06 | 2.55 | \ |
| FP64 | 1.12 | 0.52 | 0.20 | \ | \ |

We present the performance summary of platforms in Table 3, which is computed by taking the geometric mean of benchmarks and using V100 FP32 format as the normalization baseline. Among all platforms, Graphcore IPU GC200 delivers the highest FP32 performance, while A100 achieves the highest FP16 and FP64 performance.

## 3.2 BERT Operator Benchmark

We benchmark the BERT model's operators across a variety of batch sizes, sequence lengths, and the number of heads. For the Graphcore GC200 IPU platform, there are 8 configurations for the batch size: [1, 2, 4, ..., 128], and 8 configurations for (sequence length, number of head) pairs: [(128, 12), (128, 16), (128, 32), (384, 12), (384, 16), (384, 32), (512, 16), (512, 32)]. For other platforms, there are 11 configurations for the batch size: [1, 2, 4, ..., 1024], while the (sequence length, number of head) configurations remain the same as those in the IPU platform setup. Benchmark results are presented in Fig. 3.

## 3.3 2D Convolution Operator Benchmark

We benchmark operator performance on three commonly used convolutional neural networks (CNNs) with the same input image size as the ImageNet dataset [22]: ResNet-18, ResNet-50 [14], and MobileNetV2 [27]. We evaluate the CNN operators on eight batch size configurations (1, 2, 4, ..., 128) using the Graphcore GC200 IPU platform and 11 batch size configurations (1, 2, 4, ..., 1024) on other platforms. The benchmark results are presented in Fig. 4.

**Table 4: Normalized geomean hardware throughput for convolution.**

| Platform | A100 | V100 | MI100 | GC200 | SN10 |
|---|---|---|---|---|---|
| FP16 | 2.26 | 1.48 | 1.99 | 8.18 | 0.35 |
| FP32 | 1.86 | 1.00 | 1.26 | 4.60 | \ |
| FP64 | 0.26 | 0.16 | 0.14 | \ | \ |

The geometric mean benchmark results for CNNs are presented in Table 4, where the V100 FP32 format is used as the baseline for normalization. In both FP16 and FP32 formats, the Graphcore IPU GC200 platform outperforms all other platforms with at least a 2× speedup.

## 3.4 SPMM Benchmark

Graph neural network (GNN) is significant workload in the field of AI/ML, and their performance bottleneck is the sparse-matrix dense-matrix multiplication (SPMM) [30]. We evaluate the performance of SPMM on both Graphcore IPU GC200 and GPU platforms. For the IPU platform, we use PopSparse library [12] and follow the benchmarking example in [7]. PopSparse currently only supports COO format for sparse matrices and provides backpropagation on both sparse and dense inputs. For GPU platform, we use PyTorch sparse library [26] to perform SPMM, which supports two sparse matrix formats: coordinate (COO) and compressed sparse row (CSR). COO format-based PyTorch SPMM supports sparse gradient backpropagation, while CSR format does not. We also provide SPMM benchmark using PyTorch Geometric (PyG) [6], an open-source message-passing-based framework, on Nvidia GPU.

We conduct benchmarks on a selection of 100 sparse matrices sourced from the SuiteSparse library [5]. To generate additional results, we vary the second dimension (batch size) of the dense matrix. The benchmark is performed on both the GC200 IPU platform and GPU platforms, with batch sizes ranging from 8 to 4096 and 8 to 8192, respectively. Figure 5 shows the raw results.

**Table 5: Normalized geomean hardware throughput for SPMM.**

| Framework | PyG | | | Torch CSR | | | Torch COO | |
|---|---|---|---|---|---|---|---|---|
| Platform | FP16 | FP32 | FP64 | FP16 | FP32 | FP64 | FP32 | FP64 |
| A100 | 3.86 | 3.47 | 3.36 | 8.16 | 12.24 | 9.29 | 0.95 | 0.78 |
| V100 | 2.73 | 2.64 | 2.45 | 10.01 | 11.25 | 8.54 | 1.00 | 0.81 |
| MI100 | \ | \ | \ | \ | 3.14 | 2.64 | 0.35 | 0.29 |
| GC200 | \ | \ | \ | \ | \ | \ | 1.06 | \ |

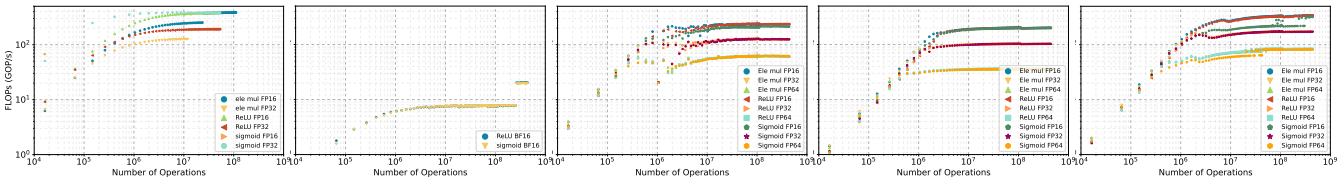
The benchmark results' geometric mean is presented in Table 5, with the V100 FP32 format as the normalization baseline. The PyTorch CSR on V100/A100 platform achieves the highest throughput for the SPMM benchmark. The PyG framework exhibits more stable performance for differentiable SPMM and supports more data formats. The Graphcore IPU GC200 platform shows great potential in SPMM tasks, with a high peak performance; however, the performance is not optimized for varying problem sizes.

## 3.5 Streaming Operator Benchmark

We benchmark streaming operators, including element-wise square and non-linear operators such as ReLU and Sigmoid, on both platforms. The Sambanova SN10 platform's compiler stack, SambaFlow, does not support the individual element-wise square operator; therefore, it is not recorded. The benchmark results of streaming operators are presented in Fig. 6.

# 4 CONCLUSION

In this study, we evaluate and compare three major types of hardware platforms for AI/ML acceleration: Graphcore IPU, Sambanova RDU, and GPU. Through our benchmarking, we have identified the potential of the Graphcore IPU for accelerating AI/ML applications such as CNNs and GNNs. Overall, our work contributes to a better understanding of the current state-of-the-art in AI/ML hardware acceleration and provides guidance for future research in this field.




**(a) Graphcore** **(b) Sambanova** **(c) MI100** **(d) V100** **(e) A100**

**Figure 6: Cross platform evaluation on element-wise operators.**

## ACKNOWLEDGEMENT

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, "ComPort: Rigorous Testing Methods to Safeguard Software Porting", under Award Number 78284. This work uses platforms supported by U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation".

## REFERENCES

[1] Nvidia Corporation. [n. d.]. CUDA C++ Programming Guide. Retrived from https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. Accessed: 2022, Oct. 30th.

[2] Nvidia Corporation. [n. d.]. NVIDIA A100 Tensor Core GPU Architecture, unprecedented acceleration at every scale. Retrived from https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf. Accessed: 2022, Oct. 30th.

[3] Nvidia Corporation. [n. d.]. Nvidia Tesla V100 GPU Architecture, The World's Most Advanced Data Center GPU. Retrived from https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf. Accessed: 2022, Oct. 30th.

[4] Sambanova System Corporation. [n. d.]. SambaFlow - SambaNova Systems. Retrived from https://sambanova.ai/wp-content/uploads/2021/04/SambaNova_SambaFlow_Datasheet_English.pdf. Accessed: 2022, Oct. 30th.

[5] Timothy A Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1–25.

[6] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[7] Graphcore. [n. d.]. Graphcore: Dynamic Sparsity. Retrived from https://github.com/graphcore/examples/tree/master/sparsity/dynamic_sparsity/tensorflow1. Accessed: 2022, Oct. 30th.

[8] Graphcore. [n. d.]. Introducing the Colossus$^{TM}$ MK2 GC200 IPU. Retrived from https://www.graphcore.ai/products/ipu. Accessed: 2022, Oct. 30th.

[9] Graphcore. [n. d.]. IPU-POD16 Direct Attach Datasheet. Retrived from https://docs.graphcore.ai/projects/ipu-pod16-datasheet/en/latest/product-description.html. Accessed: 2022, Oct. 30th.

[10] Graphcore. [n. d.]. IPU Programming Model. Retrived from https://docs.graphcore.ai/projects/ipu-programmers-guide/en/latest/programming_model.html. Accessed: 2022, Oct. 30th.

[11] Graphcore. [n. d.]. Poplar Graph Framework Software. Retrived from https://www.graphcore.ai/products/poplar. Accessed: 2022, Oct. 30th.

[12] Graphcore. [n. d.]. PopSparse Matrix Multiplication (Dynamic Pattern) on the IPU. Retrived from https://docs.graphcore.ai/projects/dynamic-sparsity/en/latest/dynamic-sparsity.html. Accessed: 2022, Oct. 30th.

[13] Reiner Hartenstein. 2001. Coarse grain reconfigurable architectures. In *Proceedings of the ASP-DAC 2001. Asia and South Pacific Design Automation Conference 2001 (Cat. No. 01EX455)*. IEEE, 564–569.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[15] AMD Inc. [n. d.]. AMD CDNA 2 Architecture. Retrived from https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf. Accessed: 2022, Oct. 30th.

[16] AMD Inc. [n. d.]. AMD CDNA Architecture. Retrived from https://www.amd.com/system/files/documents/amd-cdna-whitepaper.pdf. Accessed: 2022, Oct. 30th.

[17] AMD Inc. [n. d.]. AMD Graphics Cores Next (GCN) Architecture. Retrived from https://www.techpowerup.com/gpu-specs/docs/amd-gcn1-architecture.pdf. Accessed: 2022, Oct. 30th.

[18] AMD Inc. [n. d.]. HIP: C++ Heterogeneous-Compute Interface for Portability. Retrived from https://github.com/rocm-developer-tools/hip. Accessed: 2022, Oct. 30th.

[19] AMD Inc. [n. d.]. ROCm Documentation. Retrived from https://rocmdocs.amd.com/_/downloads/en/latest/pdf/. Accessed: 2022, Oct. 30th.

[20] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. 2019. Dissecting the graphcore ipu architecture via microbenchmarking. *arXiv preprint arXiv:1912.03413* (2019).

[21] David Koeplinger, Matthew Feldman, Raghu Prabhakar, Yaqi Zhang, Stefan Hadjis, Ruben Fiszel, Tian Zhao, Luigi Nardi, Ardavan Pedram, Christos Kozyrakis, et al. 2018. Spatial: A language and compiler for application accelerators. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 296–311.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[23] Kunle Olukotun. 2020. *Plasticine-A Universal Data Analytics Accelerator*. Technical Report. Leland Stanford Junior University Stanford United States.

[24] Raghu Prabhakar, Sumti Jairath, and Jinuk Luke Shin. 2022. SambaNova SN10 RDU: A 7nm Dataflow Architecture to Accelerate Software 2.0. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 350–352.

[25] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. 2018. Plasticine: a reconfigurable accelerator for parallel patterns. *IEEE Micro* 38, 3 (2018), 20–31.

[26] Pytorch. [n. d.]. TORCH.SPARSE. Retrived from https://pytorch.org/docs/stable/sparse.html. Accessed: 2022, Oct. 30th.

[27] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[28] Dean M Tullsen, Susan J Eggers, and Henry M Levy. 1995. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the 22nd annual international symposium on Computer architecture*. 392–403.

[29] Leslie G Valiant. 1990. A bridging model for parallel computation. *Commun. ACM* 33, 8 (1990), 103–111.

[30] Xi Xie, Hongwu Peng, Amit Hasan, Shaoyi Huang, Jiahui Zhao, Haowen Fang, Wei Zhang, Tong Geng, Omer Khan, and Caiwen Ding. 2023. Accel-GCN: High-Performance GPU Accelerator Design for Graph Convolution Networks. *arXiv preprint arXiv:2308.11825* (2023).

# FAIR Sharing of Data in Autotuning Research (Vision Paper)

Jana Hozzová
Institute of Computer Science,
Masaryk University
Brno, Czech Republic
hozzova@mail.muni.cz

Jacob O. Tørring
Department of Computer Science,
Norwegian University of Science and Technology
Trondheim, Norway
jacob.torring@ntnu.no

Ben van Werkhoven
Leiden Institute of Advanced Computer Science, Leiden University
Leiden, The Netherlands
b.van.werkhoven@liacs.leidenuniv.nl

David Střelák
Institute of Computer Science,
Masaryk University
Brno, Czech Republic
National Biotechnology Center
Madrid, Spain
373911@mail.muni.cz

Richard Vuduc
Georgia Institute of Technology
Atlanta, USA
richie@cc.gatech.edu

## ABSTRACT

Autotuning is an automated process that selects the best computer program implementation from a set of candidates to improve performance, such as execution time, when run under new circumstances, such as new hardware. The process of autotuning generates a large amount of performance data with multiple potential use cases, including reproducing results, comparing included methods, and understanding the impact of individual tuning parameters.

We propose the adoption of FAIR Principles, which stands for Findable, Accessible, Interoperable, and Reusable, to organize the guidelines for data sharing in autotuning research. The guidelines aim to lessen the burden of sharing data and provide a comprehensive checklist of recommendations for shared data. We illustrate three examples that could greatly benefit from shared autotuning data to advance the research without time- and resource-demanding data collection.

To facilitate data sharing, we have taken a community-driven approach to define a common format for the data using a JSON schema and provide scripts for their collection.

The proposed comprehensive guide for collecting and sharing performance data in autotuning research can promote further advances in the field and encourage research collaboration.

## CCS CONCEPTS

• **General and reference** → *Measurement*; **Performance**; • **Software and its engineering** → *Collaboration in software development*.

## KEYWORDS

autotuning; benchmarks; performance; measurements; open data; data sharing

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651429

**ACM Reference Format:**
Jana Hozzová, Jacob O. Tørring, Ben van Werkhoven, David Střelák, and Richard Vuduc. 2024. FAIR Sharing of Data in Autotuning Research (Vision Paper). In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651429

## 1 INTRODUCTION

Autotuning is an automated process, guided by experiments, that selects one from among a set of candidate computer program implementations to improve its execution time, energy, or another performance characteristic [2]. An *autotuner* refers to a system that implements this process and may be viewed as having two main parts: a *code generator* that can produce a space of possible implementations; and a *search mechanism* that explores this space, using models or automated benchmarking experiments, to find one that performs well. During this process, an autotuner may generate and collect a considerable amount of performance data.

These data have value for other researchers, but making them easily reusable is a complex task. As a focal point for thinking through data-sharing issues, we selected a research problem that arises in autotuning research: how to compare search methods fairly. For instance, there may be differences in how the samples were measured and the environment in which data were collected. In benchmarking search methods, it is essential to know if a data set covers the full configuration space or only some subset (as with a grid search, for instance). As another example, it might be useful to know whether or not so-called invalid configurations are included in the data set and how these can be recognized.

The field of autotuning research would benefit from a thorough discussion about how and what to collect and share so that other researchers might be able to find, access, integrate and reuse performance data. At the time of this writing, several autotuning systems are under active development, and data sharing is considered a prerequisite for better understanding which methods work best, under what scenarios, and why.

In this paper, we outline principles, guidelines, and mechanisms to reflect the questions about performance data sharing from the view of developers, practitioners, and users of autotuning systems.

It was initiated during a meeting of autotuning researchers in March 2022[1].

Moreover, in collaboration with other researchers, we devised JSON schema to define a common format for sharing the data and a script to facilitate their collection. We suggest best practices for sharing autotuning data specifically while avoiding being overly prescriptive. We derive a comprehensive checklist of requirements for shared data using the following analysis: see Checklist for FAIR Sharing of Data in Autotuning Research or repository[2] for downloadable version. We consider these guidelines, the schema, the script and the checklist to be main contributions of this paper. Together, they provide a comprehensive view on sharing data in autotuning, offer a standard format of data and help with the collection.

The value of these data lies in multiple potential use cases. These can be categorized as:

**Reproduction and Verification:**

- Reproducing results of other researchers.
- Comparing search methods [17].
- Storing "wisdom" to cache best-results in specific tuning contexts [5].

**Search Method Modifications and Code Generation:**

- Using data to drive code generation, *a la* profile-guided optimizations in compilers.
- Using data to optimize the tuning budget [11].
- Constructing models to guide tuning, either by an expert or by a machine learning method [6].

**Analysis and Insights:**

- Running and developing statistical data analysis, machine learning, and visualization methods "tuned" to autotuning.
- Understanding the impact of individual tuning parameters and their interactions.
- Developing insights into the behavior of compilers and computer architectures.
- Studying the sensitivity of programs to their inputs and characteristics.

**Curation and Benchmarking:**

- Curating of entire tuning spaces used as "ground truth".
- Curating performance with respect to various metrics, including time, energy, storage, and accuracy.
- Making leaderboards or scoreboard reporting for specific classes of problems.

The rest of the paper is organized as follows. First, we propose data sharing guidelines in the context of autotuning (see Sharing autotuning data following FAIR principles). We adopted the doctrine of Findable, Accessible, Interoperable, and Reusable digital assets, also known as the FAIR Principles, [3] to organize these guidelines [16]. Second, we outline several use cases that show how shared performance data can stimulate research in autotuning (see Use Case Examples). Third, we summarize surveyed work on scientific data sharing and reproducibility in general, as well as performance data benchmarking, collection, and sharing in particular (see Related Work). Fourth, we describe the challenges of data sharing and lay out future work that would foster the discussion of these questions and the adoption of our approach (see Challenges and Future Work).

## 2 SHARING AUTOTUNING DATA FOLLOWING FAIR PRINCIPLES

The FAIR Guiding Principles suggest that shared data will be most useful to the broader scientific community if it is findable, accessible, interoperable, and resuable [16]. We recommend how to apply these desiderata to autotuning datasets, focusing on what information should be included to enable the use cases mentioned in Introduction.

For concreteness, other researchers and we devised a JSON schema for sharing the metadata regarding the autotuning process and its results.[4] Moreover, we implemented a script that automatically collects recommended information about software and hardware involved in autotuning experiments.[5] Several autotuners under active development already export their outputs in accordance with JSON schema [12, 15]. Additionally, the BAT project [13, 14] supports exporting data from many Python-based tuners that do not natively support the format.

In this section, we define the terms we use, and then we list recommendations and guidelines by following FAIR principles.

### 2.1 Used nomenclature

In this proposal, we adopt the OpenML distinction between **raw datasets** and **run datasets**.[6] A raw dataset is an exhaustive search of a benchmark over the entirety of a searched configuration space. The dataset of an exhaustive search would contain the global optimum. In some cases, even a dataset of a partial exhaustive search of the configuration space might be useful to share.

A run dataset consists of data from runs of the search algorithm. Tuning configurations included in it represent the path of the search method, as it was going through the searched configuration space, looking for well-performing configuration. This dataset might not contain the global optimum.

In a common scenario, a user could download a raw dataset and evaluate different search techniques using it as a surrogate for real hardware. We refer to this as a **simulated run**. Each run dataset is the full experimental result for how this search would have performed on the configurations contained in the raw dataset. A run dataset could also hold the results from a **hardware run** that is not simulated.

A raw dataset should include information that could be used to set up the same environment and run the benchmarks, thereby reproducing the exhaustive search's results. For a simulated run dataset, authors could provide a DOI to the raw dataset and thus

[1] Generic Autotuning Technology for GPU Applications at the Lorentz Center, March 7–13, 2022: https://www.lorentzcenter.nl/generic-autotuning-technology-for-gpu-applications.html

[2] https://github.com/odgaard/TuningSchema/tree/T4/checklist.md

[3] Developed, coincidentally, by others during a Lorentz Center workshop in 2014 https://www.lorentzcenter.nl/jointly-designing-a-data-fairport.html

[4] https://github.com/odgaard/TuningSchema/tree/T4/metadata.py

[5] https://github.com/odgaard/TuningSchema/blob/T4

[6] https://openml.org

simplify the metadata that they need to provide. If the authors provide a hardware run dataset, there would be additional requirements to ensure that other researchers can reproduce the results.

We use the term **kernel** to refer to a unit of autotuned code. A kernel may be a CUDA kernel, the critical routine of an application, or an entire application if autotuned as a whole via, for instance, tuning compiler switches.

## 2.2 Findable

"Findability" refers to the ability to find and filter data of interest by other researchers. We strongly recommend that all datasets have DOIs to make citation possible. We mention other approaches to make datasets more available to other researchers in section Challenges and Future Work.

## 2.3 Accessible

"Accessibility" refers to the ability of other researchers to obtain a copy of the dataset. We recommend using repositories like Zenodo to host autotuning datasets. Zenodo data artifacts are citable via DOI and available for open or closed submission with versioning of datasets of 50 GB (or bigger if needed). We also recommend that authors provide contact details of the person responsible for collecting the data.

## 2.4 Interoperable

"Interoperability" means providing the information necessary to use the data in another autotuner or computing environment. Doing so implies a common data format or an otherwise completely and precisely described format and provides information related to invalid data points.

*2.4.1 Standard Data Formats.* It is a good practice to format data in standard and widely recognized formats for easy interpretation and integration. For example, JSON is recommended as it is universally recognized and easy to parse.

*2.4.2 Performance Measurement Recommendations.* For accurate performance measurements, it is necessary to report key details that align with the metrics for search method comparisons. This includes:

- Kernel experiment time, or the individual run time of kernel configurations.
- Time spent validating the output of the kernel configuration.
- Compilation time.
- Overhead details, such as the search method overhead (inclusive of model prediction time) and the autotuner's overhead.
- The timestamp of the measurement.

*2.4.3 Additional Measurements and Metrics.* It is beneficial to report supplementary measurements, such as power consumption, profiling counters, and clock frequencies, whenever possible. Derived metrics like compute performance in GFLOP/s or energy efficiency in GFLOPS/W can provide deeper insights. Importantly, if there is variability in some measurements due to factors like performance counter acquisition, it is crucial to describe the reasons and mark the data accordingly.

*2.4.4 Detailed Information on Tuning Parameters.* Tuning parameters should be described in detail. This encompasses the name, type (like int, float, string), and values or range (valid and used). Any conditions or restrictions on these parameters, especially complex ones, should be documented. It should be explicitly stated if there is a relationship between the input problem size and tuning parameters.

*2.4.5 Handling of Invalid Data Points.* Including invalid data points, especially in full configuration space explorations, provides a complete picture. Such data points should be marked based on their type, whether due to not meeting conditions, failed compilation, or computational issues during runtime.

*2.4.6 Validation.* Details about the validation of the results, including the benchmarked kernel configurations and their correctness criteria, must be provided.

*2.4.7 Miscellaneous Recommendations.* Minor details include the notation for decimal points (dot or comma). We also advise noting the timestamp of the experiment's start, as the GPU's runtime duration might influence the performance.

## 2.5 Reusable

The concept of "Reusability" in the context of dataset management extends beyond merely providing the dataset; it involves furnishing essential and supplementary information that aids in further research utilizing the dataset. This necessitates comprehensive metadata detailing both the data collection and processing methodologies.

*2.5.1 General Dataset Information.* Several key elements are integral to ensuring both clarity and traceability in dataset usage. These include providing links to associated research and data papers and offering a license recommendation, such as the Open Data Commons Open Database License (ODbL) 1.0 (available at [https://opendatacommons.org/licenses/odbl/1-0/]). Additionally, it is useful to clarify any data usage restrictions and establish clear citation guidelines for the dataset.

*2.5.2 Computational Problem Description.* Explaining the computational problem in lay terms would aid a broader understanding, particularly for those not specialized in the field. This should be complemented by an outline of common programming patterns into which the problem might fit and a clarification regarding whether the problem is predominantly memory-bound or compute-bound.

*2.5.3 Kernel Implementation Details.* Detailing the specifics of the kernel's programming aspects is important. This includes information on the source code's location and version, the programming language employed, details of the compiler and its options, the kernel's grid and thread size, and specifics about the kernel's arguments.

*2.5.4 Tuning Parameter Insights.* An in-depth exploration of the tuning parameters and their impacts is recommended. This should cover the effects of commonly used tuning parameters, details of the dataset's configuration space, and information on the dataset's run data, including the search methods and models utilized.

*2.5.5 Input Data Details.* Providing information on the dataset's inputs is suggested. This encompasses detailing the input's size and other relevant characteristics and whether this input data is included within the dataset.

*2.5.6 Data Collection and Processing.* A comprehensive dive into data management practices is necessary. This involves outlining the data acquisition methods, autotuner details, techniques used in data processing and filtering, visualization and other scripts, execution environment details, and the software and scripts employed to ensure reproducibility.

*2.5.7 Hardware Specifications.* Providing in-depth details about the hardware utilized in the dataset's creation and processing is significant to ensure reproducibility. This should include information on device details and model numbers, chipsets and memory specifics, methods for measuring power consumption, and details as recommended by the Supercomputing conference environment script[7].

*2.5.8 Environment and Execution Details.* Detailing the ecosystem surrounding the dataset makes the data actually reusable. This includes specifics on the software used, including operating systems and compilers, details of scripts and software for dataset acquisition, and information related to compilation and execution.

This comprehensive list of recommended shared information about data may be considered intimidating. Therefore, we encourage to use the script[8] we developed in collaboration with other researchers to automate the collection of bulk of these metadata. We provide an actual checklist in PDF and MD format for easier reference in the same repository and in the appendix of this paper.

## 3 USE CASE EXAMPLES

Shared performance data have many potential use cases; we listed several in section Introduction. In this section, we elaborate on three of them in greater detail. With these use cases, researchers could aim to devise more effective and faster search methods by better understanding current search methods or by deeper insight into tuning spaces.

One of the main issues related to the search in autotuning is that tuning spaces are hard to search. They are discrete, non-convex, and show little or no locality, i.e. similar configurations usually have vastly different performance. It means that many traditional approaches for search in optimization space do not perform well or they do not perform better than random search [3].

Reusing the performance data might be extremely difficult or even impossible in all use cases without all the necessary information. Our recommendations on sharing autotuning-related data have been chosen with these use cases (and more) in mind.

### 3.1 Comparing search methods

Comparing search methods and understanding which work best and under what scenarios facilitates the development and use of effective and fast search methods. However, fair comparison presents a difficult task in itself. The simplest way (although it also has its pitfalls) is to have all the search methods in one autotuner. Then, one could search for the best configuration of the same computational problem using the same input on the same hardware and see which search method finds the solution the fastest. To provide more robustness and fairness to comparison, one would repeat it many times to deal with the stochastic nature of the search, change out the input and hardware to deal with the sensitivity and possibly even change the parameters of the search method. Even if we ignore the implementation phase that might be needed, preparing and executing all of these experiments is heavily time- and resource-consuming.

Reusing performance data shared by other researchers would save a lot of time and resources, as someone else has already done the most consuming part. Clearly, a fair comparison of search methods implemented within different autotuners poses another set of challenges. It might even be impossible if the shared dataset does not include all the information needed, e.g. kernel code to ensure that we compare the same computational problem; details about performance measurements (whole experiment time vs. separately noting kernel time and compilation time and overhead time) to ensure we can account for the overhead of the search; or execution environment (hardware and software setup details) to ensure we compare experiments run on the same or comparable hardware.

Recently, Willemsen et al. [17] published a method for comparing search methods in different autotuners fairly. Our list includes all the data needed to use their comparison metric.

### 3.2 Creating models to guide tuning

One of the ways to search faster is to give it domain- and problem-specific information, for example, a performance model created either by an expert or a machine learning technique. If trained properly, it can assess the shape of the tuning space of the given computational problem and guess what configuration would be the best next step.

In order to create and train the performance model, one needs performance data above the usual time or energy spent in the kernel. Performance profiling counters, also called hardware counters, provide more reusable information. However, their collection is heavily time- and resource-consuming; kernel run with profiling turned on runs much slower than usual. Thus, reusing the data collected by others would save many resources. Apart from profiling counters, detailed information about the tuning space and computing environment is required.

Data from previous runs on other hardware or input have been used to guide the tuning in [6]. Filipovič et al. gathered all profiling counters themselves and explicitly noted how demanding it was. They shared the data in [9]. Machine learning method based on statistical analysis guides the profiling also in [10].

### 3.3 Analyzing data to gain insight

Gaining deeper insight into tuning spaces would help researchers make more informed decisions while developing search methods or creating performance models to guide the tuning. Via statistical analysis, they can inspect the interactions between tuning parameters, look at input sensitivity and learn the intricate details

[7] https://submissions.supercomputing.org/?page=SampleForm&id=PaperArtifactDescriptionArticleEvaluationADAEAppendix&site=sc21
[8] https://github.com/odgaard/TuningSchema/blob/T4

about hardware and programming models beyond often poorly documented behaviour.

Running all the experiments "just" for statistical analysis seems wasteful and worthless. By using shared data, one can build upon the work of others and move the research further.

## 4 RELATED WORK

Several related initiatives have been created for or could be reasonably used towards sharing data in autotuning research. For instance, there are many machine learning and data science competition platforms, such as Kaggle[9] or OpenML[10], that allow researchers to publish or share datasets. While the data sharing principles we have proposed could be implemented using any of these platforms, our guidelines for autotuning go beyond the typical use case of these platforms by providing specific recommendations with regard to what metadata to collect and how datasets can be made FAIR.

Several computer science conferences and journals are currently trying to improve the reproducibility and replicability of results by implementing artifact description and evaluation initiatives. We have drawn inspiration from the questionnaire used for artifact evaluations in SuperComputing[11] and specialized them to cover information required to enable FAIR sharing of autotuning performance measurement data.

In terms of autotuning research specifically, a set of initiatives by Grigori Fursin under the names of cTuning[12], Collective Mind [8], Collective Knowledge (CK) [7], and work of Cho et al. [4] are probably the most closely related works. The cTuning initiative started with a similar goal: to enable or bootstrap sharing of performance data within the autotuning research community. Collective Mind aimed to crowd-source such autotuning performance data for many different computing hardware systems. It seems these systems have inspired Collective Knowledge, a system for automating research actions similar to CodaLab.[13] It appears that cTuning can only be used through the Collective Knowledge workflow. It is unclear how, where, and in what form the data or metadata is stored. The main contribution of the initiative appears to be in automating the experimentation workflow through the Collective Knowledge system.

Finally, Cho et al. have proposed a JSON database to leverage historical data for Bayesian optimization and provide CK for metadata collection [4].[14]. They also propose a website where users can upload their results.[15] However, this repository is specific to their autotuner GPTune (which is limited to integer, real, and categorical types of parameters and the Bayesian optimization process), whereas what we are proposing tries to be more general and inclusive of other autotuning frameworks. The developers of GPTune also collect additional metadata, such as software version, machine information and evaluation data. These are publicly available, with over 14 thousand evaluations spanning 23 problems. Capitalizing on these data-sharing opportunities is a central goal of our more general proposal.

## 5 CHALLENGES AND FUTURE WORK

Many researchers may view data sharing as a hardly feasible task - it is undoubtedly a challenge. We have addressed a few perceived barriers in this paper; several remain for future work.

The main challenges of data collection include that researchers do not know *what* to collect and *how* to collect data without too much effort. Data sharing brings additional worries: *where* to share and *what* to share so that the data can be reused. Moreover, data collection and sharing are usually not at the center of the researcher's attention; they are just a byproduct of the ongoing research. Therefore, anything that makes data collection and sharing quicker and easier can sway the decision to make public or to keep private.

In our guidelines, we address the question about *what* data should be collected, both in terms of actual, primary data and additional descriptions of the environment and processes. All these suggestions have been made with a focus on the future reusability of data. Having a checklist gives researchers a solid starting point, ensuring they do not omit anything critically important.

The question of *how* to collect data can be divided into three parts: primary data (measurements and tuning space details), environment data (hardware information, code location) and process data (details about the computational problem, search method and data processing). Collecting the measurements and details about tuning space in our proposed JSON format is automatic in some autotuning frameworks. So far, KTT [12] and KernelTuner [15] export data in this format, and BAT project [13, 14] can export data from several Python-based tuners, most notably OpenTuner [1]. For the environment data, we offer the script that automatically collects most of them. The last part is the most time-consuming; details about methods and processes need to be written down. However, this effort pays off later, as these descriptions most probably should also appear in the research article; they are not relevant only to data sharing.

We address the question of *where* to share the data by suggesting both the repository and licence. To make autotuning data more readily findable, we foresee a website that could serve as a primary entry point. For those creating data, it should provide an easy way to submit links to their datasets and facilitate data sharing. For those seeking data, it should offer basic filtering on the metadata properties of the datasets to help look up by criteria of specific interest in autotuning, such as what hardware or performance metrics were used. For example, EOSC initiative[16] aims to provide a venue for such websites.

We realize that even with the guidelines, a checklist and scripts at our disposal, the feat of data sharing might seem unreachable. A sample dataset and a sample description of data that would abide by our guidelines ease the hurdle and make it conceivable to undertake. We have created a simple example of shared data https://zenodo.org/records/7212426. It contains only automatically generated data, no detailed descriptions of all processes. Nevertheless, it can illustrate how the shared data generated by our scripts and schemas look like.

[9] https://www.kaggle.com/
[10] https://www.openml.org/
[11] https://submissions.supercomputing.org/?page=SampleForm&id=PaperArtifactDescriptionArticleEvaluationADAEAppendix&site=sc21
[12] https://ctuning.org/
[13] https://worksheets.codalab.org/
[14] https://github.com/gptune/CK-GPTune
[15] https://gptune.lbl.gov
[16] https://eosc-portal.eu/

## 6 CONCLUSION

We have formulated several recommendations and guidelines to enable sharing FAIR data with the autotuning research community based on twelve foreseen use cases. Based on the FAIR principles (Findable, Accessible, Interoperable, and Reusable), we recommend specific actions the autotuning community can take, including what metadata to collect to sustain the use cases and ultimately make the autotuning data FAIR. We hope that the community's refinement and ultimate adoption of this proposal will help address data sharing issues in autotuning. Indeed, we strongly believe such data have high intrinsic value, not just within the autotuning community but also for improving our overall understanding of computer system performance.

## ACKNOWLEDGMENTS

The CORTEX project has received funding from the Dutch Research Council (NWO) in the framework of the NWA-ORC Call (file number NWA.1160.18.316). The *Generic Autotuning Technology for GPU Applications* workshop has received funding from the Lorentz Center, Netherlands eScience Center, and the CORTEX consortium.

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ LM2018140.

This research work was also funded by the European Commision – NextGenerationEU(Regulation 2020/2094), through CSIC´s Global Health Platform (PTI Salud Global).

## REFERENCES

[1] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. 2014. Opentuner: an extensible framework for program autotuning. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. Edmonton, Canada, (Aug. 2014). http://groups.csail.mit.edu/commit/papers/2014/ansel-pact14-opentuner.pdf.

[2] Prasanna Balaprakash, Jack Dongarra, Todd Gamblin, Mary Hall, Jeffrey K Hollingsworth, Boyana Norris, and Richard Vuduc. 2018. Autotuning in high-performance computing applications. *Proceedings of the IEEE*, 106, 11, 2068–2083.

[3] Prasanna Balaprakash, Stefan M. Wild, and Paul D. Hovland. 2011. Can search algorithms save large-scale automatic performance tuning? *Procedia Computer Science*. Proceedings of the International Conference on Computational Science, ICCS 2011 4, (Jan. 1, 2011), 2136–2145. DOI: 10.1016/j.procs.2011.04.234.

[4] Younghyun Cho, James W Demmel, Xiaoye S Li, Yang Liu, and Hengrui Luo. 2021. Enhancing autotuning capability with a history database. In *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 249–257.

[5] FFTW.org. 2023. Words of wisdom - saving plans. Web page. (2023). Retrieved Feb. 8, 2023 from https://www.fftw.org/fftw3_doc/Words-of-Wisdom_002dSaving-Plans.html#Words-of-Wisdom_002dSaving-Plans.

[6] Jiří Filipovič, Jana Hozzová, Amin Nezarat, Jaroslav Ol'ha, and Filip Petrovič. 2022. Using hardware performance counters to speed up autotuning convergence on GPUs. *Journal of Parallel and Distributed Computing*, 160, (Feb. 1, 2022), 16–35. DOI: 10.1016/j.jpdc.2021.10.003.

[7] Grigori Fursin. 2021. Collective knowledge: organizing research projects as a database of reusable components and portable workflows with common interfaces. *Philosophical Transactions of the Royal Society A*, 379, 2197, 20200211.

[8] Grigori Fursin. 2013. Tutorial at hpsc 2013 at ntu, taiwan: collective mind: novel methodology, framework and repository to crowd-source auto-tuning. In *HPSC-Conference on Advanced Topics and Auto Tuning in High Performance and Scientific Computing-2013*.

[9] Jana Hozzová, Jiří Filipovič, Amin Nezarat, Jaroslav Ol'ha, and Filip Petrovič. 2021. Searching CUDA code autotuning spaces with hardware performance counters: data from benchmarks running on various GPU architectures. *Data in Brief*, 39, (Dec. 1, 2021), 107631. DOI: 10.1016/j.dib.2021.107631.

[10] Edward Hutter and Edgar Solomonik. 2021. Accelerating Distributed-Memory Autotuning via Statistical Analysis of Execution Paths. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). (May 2021), 46–57. DOI: 10.1109/IPDPS49936.2021.00014.

[11] Jaroslav Olha, Jana Hozzová, Matej Antol, and Jiří Filipovič. 2024. Estimating resource budgets to ensure autotuning efficiency. *Journal of Parallel and Distributed Computing*. Submitted. http://dx.doi.org/10.2139/ssrn.4661862.

[12] Filip Petrovič, David Střelák, Jana Hozzová, Jaroslav Ol'ha, Richard Trembecký, Siegfried Benkner, and Jiří Filipovič. 2020. A benchmark set of highly-efficient cuda and opencl kernels and its dynamic autotuning with kernel tuning toolkit. *Future Generation Computer Systems*, 108, 161–177.

[13] Ingunn Sund, Knut A. Kirkhorn, Jacob O. Tørring, and Anne C. Elster. 2021. BAT: a benchmark suite for AutoTuners. *Norsk IKT-konferanse for forskning og utdanning*, 1, (Nov. 14, 2021), 44–57. Number: 1. Retrieved Dec. 10, 2021 from https://ojs.bibsys.no/index.php/NIK/article/view/915.

[14] Jacob O. Tørring, Ben van Werkhoven, Filip Petrovič, Floris-Jan Willemsen, Jiří Filipovič, and Anne C. Elster. 2024. Towards a Benchmarking Suite for Kernel Tuners. *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Submitted.

[15] Ben van Werkhoven. 2019. Kernel tuner: a search-optimizing gpu code auto-tuner. *Future Generation Computer Systems*, 90, 347–358. DOI: https://doi.org/10.1016/j.future.2018.08.004.

[16] Mark D. Wilkinson et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3, 160018, (Mar. 2016). DOI: 10.1038/sdata.2016.18.

[17] Floris-Jan Willemsen, Richard Schoonhoven, Jiří Filipovič, Jacob O. Tørring, Rob van Nieuwpoort, and Ben van Werkhoven. 2024. A methodology for comparing optimization algorithms for auto-tuning. *Future Generation Computer Systems*. Submitted.

# A CHECKLIST FOR FAIR SHARING OF DATA IN AUTOTUNING RESEARCH

This appendix contains a checklist of recommended information to share. Information that gets automatically collected by using our scripts or those that are present in our proposed JSON schemas available in repository https://github.com/odgaard/TuningSchema/blob/T4/ is marked by ⊡.

- General information
  - ☐ name of the dataset for easier future reference
  - ☐ DOI and link to repository
  - ☐ contact information to authors
  - ☐ how to cite
  - ☐ licence and usage restrictions
  - ☐ link to related papers
- Measurements
  - ⊡ kernel experiment time
  - ⊡ validation time
  - ⊡ compilation time
  - ⊡ overhead details (search method overhead, autotuner overhead, model overhead)
  - ⊡ timestamp
  - ⊡ if possible, additional measurements, such as power consumption, profiling counters or clock frequencies
- Tuning space
  - ⊡ names and types of tuning parameters
  - ⊡ values or ranges of tuning parameters
  - ⊡ conditions of tuning parameters
  - ☐ details about how different types of invalid data points are handled
  - ⊡ details about how the results are validated
  - ☐ description of the effects of tuning parameters
  - ☐ details about search space, i.e. raw dataset or run dataset
- Computational problem and its implementation
  - ☐ explanation for non-experts
  - ☐ common programming patterns it fits into
  - ☐ memory- or compute-bound
  - ☐ source code location and version
  - ⊡ programming language used
  - ⊡ grid and thread size
  - ⊡ kernel argument details
- Search method and models
  - ⊡ hyperparameters of the search method
  - ⊡ budget
  - ⊡ performance metric and optimization objective function
  - ☐ details about how models were created and trained
- Environment and execution
  - Input data
    - ☐ size and other relevant characteristics
    - ☐ whether it is included within the dataset
  - Hardware
    - ⊡ details about the device and the model
    - ⊡ chipsets and memory specifics
    - ☐ details about how power consumption is measured
    - ⊡ details provided by the recommended Supercomputing conference environment script
  - Software
    - ⊡ software specifics, OS and compilers
    - ⊡ details about compilation
    - ⊡ details about execution environment
  - Data processing
    - ☐ details about how data were acquired
    - ☐ details about how the autotuner was set and executed
    - ☐ details about data processing and filtering
    - ☐ if relevant, details about analysis and visualization
    - ☐ software and scripts used for dataset acquisition, processing, analysis and visualization

# Mastering Computer Vision Inference Frameworks

Pierrick Pochelu
University of Luxembourg FSTM-DCS
Luxembourg
pierrick.pochelu@uni.lu

Oscar Castro-Lopez
University of Luxembourg FSTM-DCS
Luxembourg
oscar.castro@uni.lu

## ABSTRACT

In this paper, we present a comprehensive empirical study to evaluate four prominent Computer Vision inference frameworks. Our goal is to shed light on their strengths and weaknesses and provide valuable insights into the challenges of selecting the right inference framework for diverse situations. Additionally, we discuss the potential room for improvement to accelerate inference computing efficiency.

## CCS CONCEPTS

• **Software and its engineering**; • **Computing methodologies** → **Computer vision**;

## KEYWORDS

software performance, neural networks, inference

**ACM Reference Format:**
Pierrick Pochelu and Oscar Castro-Lopez. 2024. Mastering Computer Vision Inference Frameworks. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3629527.3651430

## 1 INTRODUCTION

In the field of deep learning, the deployment of trained neural networks to make predictions, a process also known as inference, is the pivotal moment where these models provide value in applications. Inference deep learning frameworks play a central role in this process by translating the mathematical representation of the neural network into low-level code optimized for specific hardware platforms. These inference frameworks employ a diverse range of optimizations aimed at significantly improving the computational speed of neural network predictions.

While the training phase of deep neural networks is generally a time-bounded computing process with a fixed number of epochs or batches, the inference phase often involves long-term deployment. This is why pursuing lower prediction time has been recognized as a strategically significant endeavor for reducing the financial cost of computing infrastructure [6] and greener computing [5].

Various benchmarks have emerged to assess the speed of deep neural networks, they often focus on the performance of specific operators like matrix multiplication and convolution. These microbenchmarks, while informative, fall short of comprehensively evaluating the intricate complexities of modern neural networks [19]. Additionally, benchmarks like Dawn Bench [3] have highlighted that neural networks frequently under-utilize computing cores due to memory transfer bottlenecks. Although studies such as MLPerf Inference [15] and ML Bench [13] provide comprehensive assessments of various inference applications, frameworks, and hardware, they generally lack in-depth analysis of the comparison between different inference frameworks and software settings. This paper aims to address this gap by providing fresh perspectives to steer the future development of inference technology and set of tools for reproducibility.

This paper explores the inference performance exhibited by the inference frameworks such as TensorRT [4], ONNX-runtime [16], OpenVINO [7], LLVM MLIR [11], TVM [2]. These inference frameworks employ a diverse range of optimizations aimed at significantly improving the computational speed of neural network inference. Additionally, we also test/compare the performance of Tensorflow XLA [12] which, unlike the aforementioned inference frameworks, is a software environment that applies optimizations for both the training and inference phases.

With the inference frameworks we benchmark different convolutional neural network (CNN) architectures used in computer vision tasks selected for their diversified neural topology: Resnet50[9] VGG19[20] and DenseNet201[10]. For each framework, we collected over 80 data points, encompassing metrics such as prediction throughput (predictions per second), loading time, memory consumption, and power consumption on both GPU and CPU hardware configurations. The code and additional plots are linked in the GitHub repository at the end of the conclusion.

The structure of this paper is as follows. In Section 2, we discuss inference frameworks state-of-art and their optimization techniques. In Section 3, we elaborate on the settings used. In Section 4, we present the experimental results with different metrics. In Section 5, we provide key insights by summarizing the lessons learned from the experiments. Finally, in Section 6 we conclude by showing the importance of this work direction, future work, and GitHub links.

## 2 POST-TRAINING REPRESENTATION AND OPTIMIZATION

Post-training representation and optimization serve as a critical bridge between model training and efficient inference deployment.

The optimization is generally done in two steps, high-level and low-level optimizations. While high-level optimizations focus on algorithmic and architectural enhancements, low-level optimizations delve into the intricacies of code generation and hardware-specific

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651430

performance tuning. The synergy between these optimization layers empowers deep learning frameworks to deliver both accuracy and speed in real-world applications. However, different inference frameworks have different implementations.

One fundamental high-level optimization strategy is known as "fusing," a technique that consolidates multiple operations into a single kernel launch. For example, sequences of convolutions can be merged into one single convolution [1]. Fusing offers several advantages, including the elimination of slow intermediate tensor storage, improved cache utilization, and the removal of synchronization barriers between operations. Notably, these optimization methods, such as XLA [12], not only accelerate inference but can also benefit the training phase. Moreover, certain fusing operations involve mathematical simplifications, like combining adjacent convolution layers together without lowering the accuracy of the model [8].

Additional high-level optimizations encompass techniques such as constant-folding and static memory planning. Constant-folding evaluates constant expressions before executing the program and static memory planning creates a plan to reuse ahead and use intermediate tensor buffers.

On the other end, low-level optimization entails converting the computational graph into highly optimized low-level code tailored for specific hardware platforms. Drawing upon decades of compilation expertise, this optimization category encompasses subexpression elimination, vectorization, loop ordering, tiling, unrolling, threading patterns, and memory caching/reusing strategies. Notably, the field benefits from two noteworthy low-level compilers and optimizers, namely TVM [2] and LLVM MLIR [11], both enriched with tensor types, which enable efficient code generation and execution.

# 3 EXPERIMENTAL SETTINGS

## 3.1 Evaluated Neural Networks

To assess the performance of different inference frameworks, we utilize a diverse set of convolutional neural networks, including VGG19, ResNet50, DenseNet201, and EfficientNetB0. Overall, these networks offer a range of characteristics, such as varying depths, widths (parametric ratio to the number of layers), and densities, providing a comprehensive evaluation of the frameworks' capabilities. The details of these architectures are summarized in Table 1.

**Table 1: Summary of the CNNs architectures used in the benchmark.**

| | $\frac{\#param.}{\#layers}$ | #layers | #jumps | #param. Jump type |
|---|---|---|---|---|
| VGG19 | 7.26M | 19 | 0 | 138M No jumps |
| ResNet50 | 0.52M | 50 | 16 | 26M Additions |
| DensetNet201 | 0.1M | 201 | 98 | 20M Concatenations |
| EfficientNetB0 | 0.06M | 89 | 25 | 5.3M Mult. and Add. |

## 3.2 Evaluated Machine Specifications

Our benchmarking experiments are conducted on two distinct machines, namely Machine A and B. Machine A is equipped with Tesla V100 SXM2 GPUs and its CPU is a dual-socket Intel(R) Xeon(R) CPU E5-2698 v4. Whereas, Machine B has NVIDIA Amper A100 PCI-E GPUs and its CPU is a single-socket AMD EPYC 7F52. Table 2 presents the full details for both machines.

For the Software Stack, we maintain consistent software versions across both machines. Our assessment involves popular inference frameworks, including TensorFlow 2.6, TensorRT 8.0, ONNX-runtime 1.10, and OpenVINO 2021. To facilitate the benchmarking process, we utilize neural network converters such as tf2onnx 1.9.3 and LLVM 14.0.

Machine A operates on Ubuntu with Python 3.9, while Machine B runs on CentOS with Python 3.8. It's worth noting that specialized real-time operating systems may enhance latency determinism and speed but could potentially impact throughput negatively. We also explored MLIR (onnx-mlir 0.2 framework [11]) on Machine B; however, it only supports ResNet50. In all our benchmarks, Tensorflow benefits from acceleration via XLA [12] (Accelerated Linear Algebra).

**Table 2: Summary of the characteristics of the machines used in the benchmark.**

| Feature | Machine A | Machine B |
|---|---|---|
| **GPU model** | Tesla V100 SXM2 | Amper A100 PCIE |
| **GPU # of cores** | 5,120 | 6,912 |
| **GPU clock speed** | 1,312MHz-1,530MHz | 765MHz-1,310MHz |
| **GPU memory** | 16GB | 40GB |
| **GPU board cons.** | 300 watts | 250 watts |
| **CPU model** | Intel XEON E5-2698 v4 | AMD EPYC 7f52 |
| **CPU # of cores** | 80 | 16 |
| **CPU clock speed** | 1.2GHz-3.6GHz | 2.5GHz-3.5GHz |
| **CPU memory** | 512GB | 256GB |
| **OS** | Ubuntu | CentOS |
| **Python version** | 3.9 | 3.8 |

## 3.3 Graph Optimization Settings

To discuss optimization settings and the effect of each performance we discuss the obtained speed up by ablation (starting from the mostly well optimized settings and discussing the impact of changing a specific setting).

- **Tensorflow XLA**: We freeze the computational graph, which means all weights are stored in read-only memory. We enable the "`optimize_for_inference_lib`" optimizer, although it doesn't significantly impact performance. XLA is enabled on the GPU because disabling it reduces speed by 15%. However, enabling XLA multiplies initialization time by a factor of six on the range of neural networks. We do not observe a performance gain from enabling XLA on the CPU. Therefore, it remains disabled.
- **ONNX-RT** (ONNX-runtime) [16]: We enable caching, as disabling it reduces speed by approximately 3%. We also enable maximum graph-level optimization, and using the default optimization settings reduces speed by 8%.
- **OpenVINO** We enabled two settings: "NCHW" and convolution fusing. NCHW stands for batch (N), channels (C),

height (H), and width (W). NCHW is a data format, a way to represent a tensor in memory and all input images are converted into this format for this framework. In a nutshell, convolution fusing combines or merges multiple convolutional operations within a CNN model into a single, more efficient operation. When convolutional fusing is disabled the execution speed is reduced by 9%. We disabled concatenation optimization because there was no significant gain in any experiment. In fact, with Densenet201 concatenation optimization reduced the performance by 1%.
- **MLIR** [11] (Multi-Level Intermediate Representation from the LLVM project): MLIR is still under development, but we could compile the Resnet50 graph with the "-O3" optimization level for performance. This optimization level represents the highest level of optimization provided by the compiler.

These settings have been carefully configured to ensure optimal performance for each framework, enabling a fair and informative comparison of their capabilities.

# 4 EXPERIMENTAL RESULTS

After the training phase, the importance of performance metrics can significantly differ depending on the application at hand. The prioritization of specific metrics over others is deeply influenced by the unique requirements of each use case. In this study, we conduct assessments focusing on prediction speed, memory utilization, power consumption, and model loading time.

## 4.1 Prediction Speed

Our assessment of prediction speed encompasses three key scenarios, each measured differently:

- Batch Applications: In scenarios where neural networks predict a substantial workload of data samples, optimizing the batch size becomes critical to maximize predictions per second, referred to as throughput. Throughput is quantified as the number of predictions per second.
- Data Flow Applications: For use cases involving sequential data sample prediction, such as real-time embedded systems and Markov Decision Processes in deep reinforcement learning, we evaluate latency, represented as the number of milliseconds required for a single prediction.
- Irregular Batch Applications: In situations where data samples arrive irregularly, such as web services serving multiple client requests, a dynamic batch computing approach is essential. Here, we carefully balance latency for responsiveness and throughput when the service faces heavy request loads.

Figure 1 and 2 show the throughput results for machines A and B, respectively. Each figure has results for the inference frameworks by varying the batch size and the model. Notice the vertical axis is log2-scale. Please note that we have opted not to display latencies for predicting a single data point (data flow applications) since these results exhibited a strong correlation with batch size 1. In cases where there are no bars in the figure, it indicates out-of-memory issues (e.g., VGG with TensorRT) or compilation errors (e.g., EfficientNet with OpenVINO).

## 4.2 Memory Consumption

Optimizing memory usage within the inference framework unlocks various GPU utilization possibilities, effectively reducing the need for investments in multi-GPU configurations and minimizing their associated power demands. The diverse memory consumption scenarios encompass serving larger models, enhancing accuracy through the management of asynchronous ensembles [14], and efficiently handling independent applications [18]. Figure 3 shows the memory consumption of an ensemble model, quantified as the combined storage occupied by the neural network on disk and the current batch of features propagating through the layers. The ensemble results from diverse topologies: VGG19 has wider convolutions, Resnet50 is deeper, and Densenet201 includes numerous jumps between layers.

## 4.3 Power consumption

Figure 4 presents the power consumption (Watt/sec.) of different inference frameworks ("TRT" for TensorRT, "TF" for Tensorflow, "ORT" for ONNX-RT) with batch sizes 1, 32, and 128. The model running is the ensemble of VGG19, Resnet50, and DenseNet201.

We use the following command to estimate GPU power draw:

```
$ nvidia-smi -i $GPUID --format=csv --query-
    ↪ gpu=power.draw --loop-ms=3000
```

Where $GPUID is the corresponding GPU identifier hosting the neural network.

It's worth noting that the Relative Standard Deviation (RSD) of instantaneous power consumption (watt) can be high, approximately around 20%. The oscillations observed in the estimated GPU consumption are attributed to a combination of factors including instruction flows, dynamic voltage and frequency, temperature regulation, and measurement error. This underscores the significance of averaging power consumption over multiple sampling to obtain a more representative value.

Power consumption is a pivotal metric, with implications for ecological sustainability, energy costs, and thermal management. We express power consumption in terms of watt-seconds required by the inference system to predict a fixed quantity of data samples, denoted as $D$. The actual value of power consumption is influenced by the specific neural network, chosen inference engine, and batch size. These measurements can be further converted into equivalent units such as CO2 emissions, energy expenses, or thermal dissipation.

Equation 1 is used to describe power consumption denoted as $E$. The variable $D$ stands for the quantity of data samples. Whereas variable $W$ represents the mean instantaneous power consumption in watts during the prediction period $T$ in seconds.

$$E = D \times W \times T \tag{1}$$

Equation 1 provides insights into the relationship between the throughput of an inference system and its power consumption. It appears that the relationship adheres to a power law $y = \alpha x^{\beta}$, with $\alpha$ and $\beta$ coefficients for different GPU generations.

The measurement teaches us two lessons for sustainable computing. First, model speed and power consumption are linked in a predictable way, their correlation is above 0.95. Second, the power

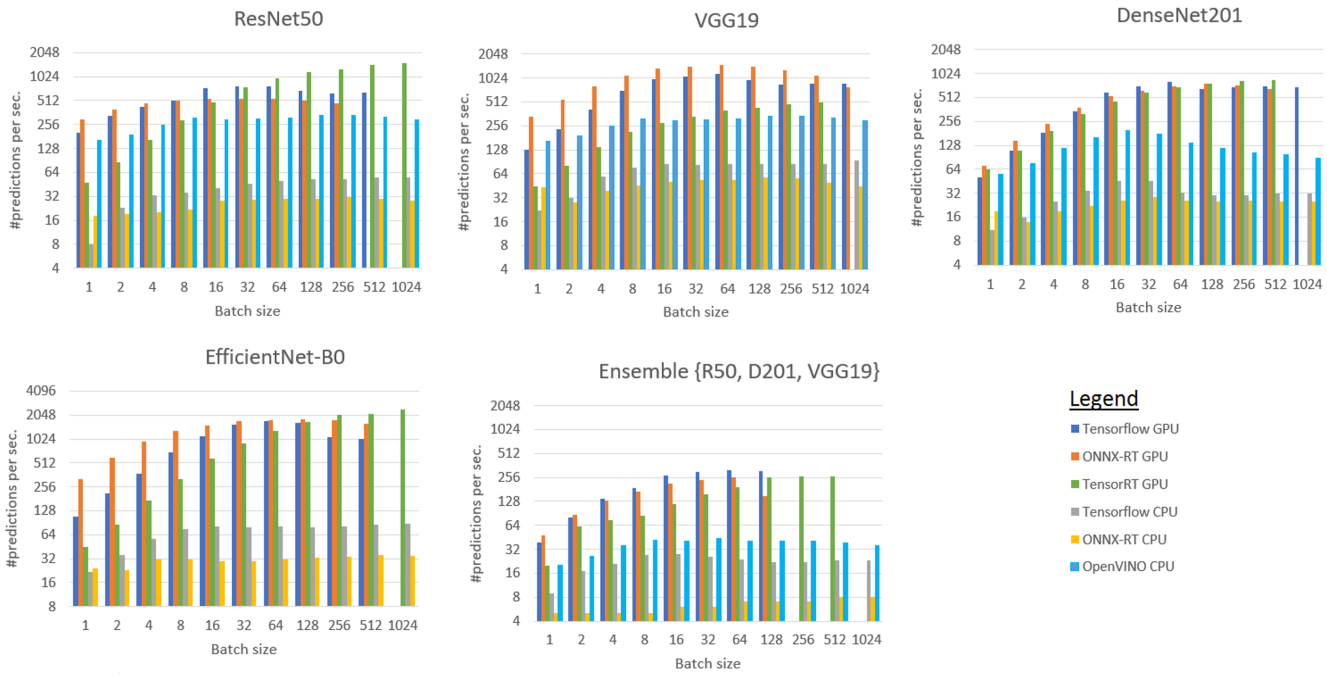



**Figure 1: Machine A. Inference framework comparison with different batch size values.**

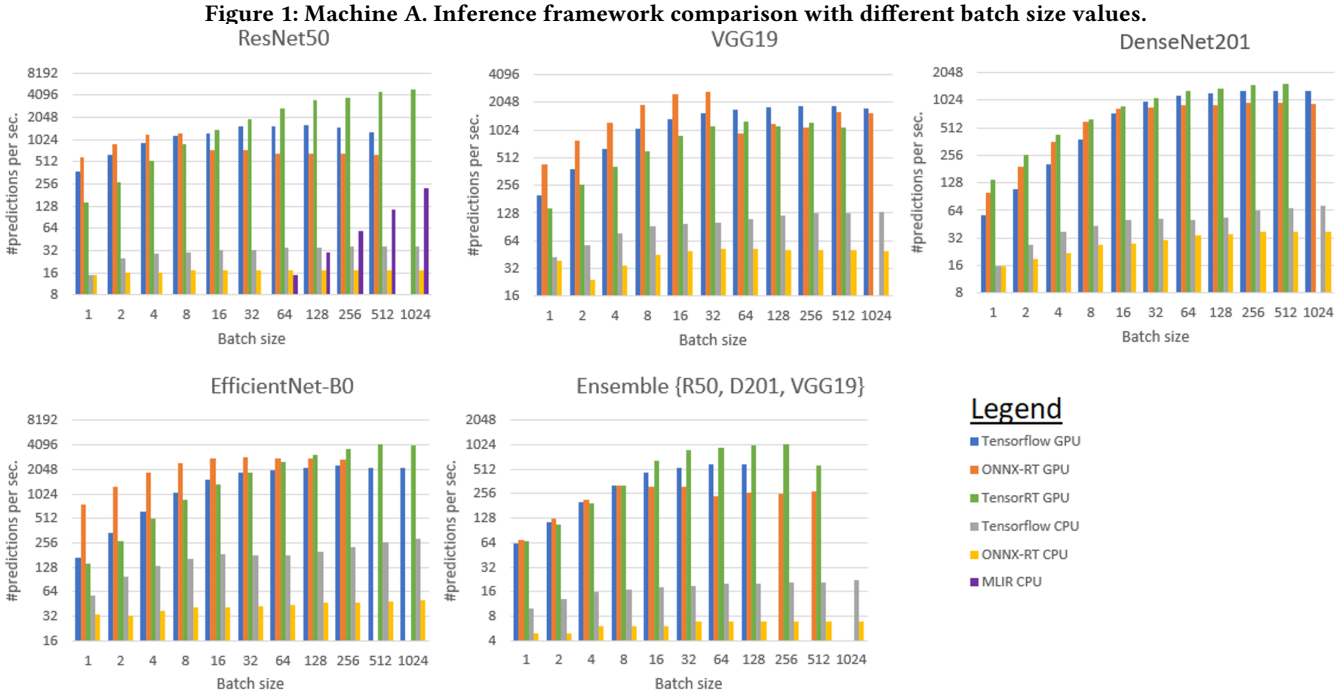



**Figure 2: Machine B. Inference framework comparison with different batch size values.**

law curvature shows us that to a certain degree, improving the model parallelism may not reduce significantly the power consumption. This reduction in the trend can be interpreted like the following: maximizing cores utilization reduces computing time ($T$) but increases instantaneous power consumption ($W$), less efficient internal parallelism keeps the cores idle which takes more computing time but consumes less power.

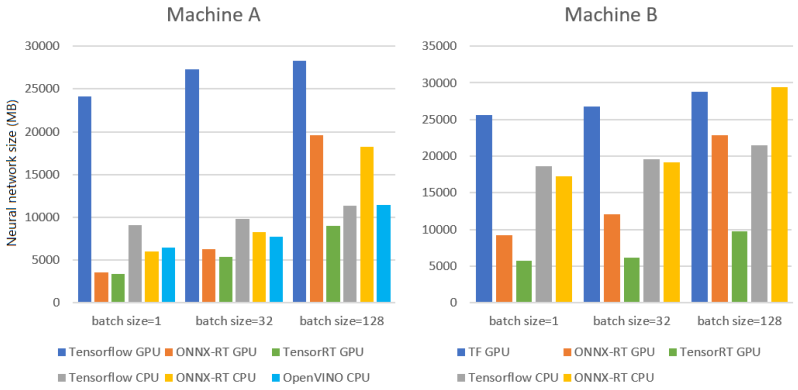



**Figure 3: Memory consumption of the model's ensemble with different frameworks varying the batch size.**

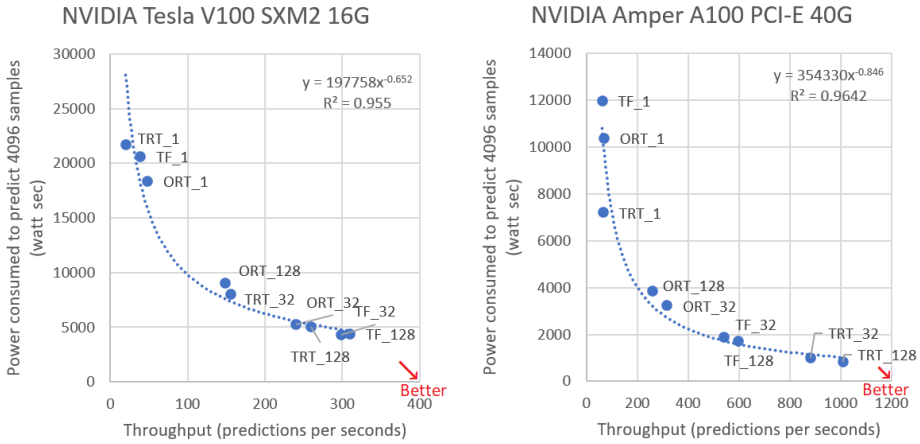



**Figure 4: Plot of the throughput and power consumption with the ensemble model varying the batch size.**

## 4.4 Loading Time

Loading time refers to the duration from the neural network's representation on disk to its readiness in memory for making predictions. This metric is of particular interest in applications requiring rapid service setup to accommodate peak demand periods, such as elastic cloud services [17]. In all our benchmarks, we enable or implement caching systems to measure loading times.

The loading times for the ensemble model (containing VGG19, Resnet50, and Densenet121) are displayed in Table 3. The data is organized by machine and processing unit (CPU/GPU), with faster loading times listed before slower ones for clarity.

TensorFlow with XLA optimization exhibits slower loading times due to the current absence of caching for optimized graphs on disk. However, disabling XLA optimization reduces loading time by a factor of 6 but it keeps staying the slowest inference framework to load the model.

**Table 3: Results of loading the models categorized by machine and processing units.**

| Machine | Device | Framework | Time (seconds) |
|---|---|---|---|
| A | CPU | OpenVINO | 3.4 |
| | | TensorFlow | 8.7 |
| | | ONNX-RT | 8.7 |
| | GPU | TensorRT | 3.6 |
| | | ONNX-RT | 5.7 |
| | | Tensorflow XLA | 42.5 |
| B | CPU | ONNX-RT | 0.5 |
| | | Tensorflow | 2.9 |
| | GPU | ONNX-RT | 2.1 |
| | | TensorRT | 3.9 |
| | | Tensorflow XLA | 29.2 |

## 5 KEY TAKEAWAYS

In this section, we offer valuable insights derived from our benchmarking outcomes. These insights are designed to aid deep learning practitioners in selecting the most suitable inference framework for their specific requirements.

The performance of GPU-based frameworks, including ONNX-RT, TensorRT, and TensorFlow with XLA, had results in response to different scenarios (batch size values and model architecture). For low-latency and sporadic request scenarios, ONNX-RT had the best results. TensorRT shines in high-throughput scenarios with larger batch sizes. TensorFlow optimized with XLA shows the good speed with high-density networks (i.e. VGG19).

For CPU inference, Intel OpenVINO consistently outperformed other CPU-based frameworks, making it a strong choice for CPU-centric deployments. MLIR shows promise and should be considered for future use as it continues to mature.

It is widely acknowledged by deep learning practitioners that optimizing the batch size for speed's sake is an empirical process. A "sufficiently large" batch size value allows inference frameworks to harness the full power of a particular processing unit. However, if the batch size value is too large, it can result in cache memory issues, potentially slowing down execution. It is worth noting that some inference frameworks have specific constraints on tensor shapes (such as TensorRT with VGG19 and batch size 1024). Furthermore, very large tensor shapes may lead to memory crashes due to indexing element errors.

In terms of GPU memory usage, TensorFlow XLA GPU consumes significantly more memory than other technologies but the gap is slightly reduced when the batch size increases. Conversely, TensorRT stands out for having the lowest overall memory footprint. For CPU memory usage, Tensorflow XLA exhibits the largest memory footprint, particularly noticeable with batch sizes 1 and 32. However, with a batch size of 128, ONNX-RT for CPU experiences a considerable increase in memory usage. The most memory-efficient option for CPU is OpenVINO.

With the power consumption metric, we provide a link between computing speed and power consumption for a fixed amount of data samples. This leads us to the conclusion that optimizing the computing time reduces the power consumption up to a certain extent.

## 6 CONCLUSION AND FUTURE DIRECTIONS

We benchmark four deep learning inference frameworks: TensorRT, ONNX-runtime, OpenVINO, and LLVM MLIR and a diverse array of neural network architectures and configurations. Some inference frameworks are still missing such as TVM and will be introduced later in our repo.

Our study has yielded valuable insights into the domain of machine learning inference optimization. We learned that selecting the ideal inference framework from the multitude of options available can be a daunting task. In addition to that, our findings underscore the importance of aligning the specific choice with the final application requirements and hardware environment. Therefore fast experimentation of the application under different settings is desirable to optimize it, this is what we published in our GitHub repo.

Looking ahead, more in-depth analysis will lead our future explorations in the design of inference development tools and inference systems. We anticipate that our work will inspire research and innovation leading to more efficient and effective machine learning solutions for making easier the development and deployment of optimized neural networks.

Supplementary materials, including the GitHub repository link and full-resolution figures, will be provided upon acceptance for the double-blind reviewing process.

## REFERENCES


[1] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783725

[2] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, USA, 579–594.

[3] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2019. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. *SIGOPS Oper. Syst. Rev.* 53, 1 (jul 2019), 14–25. https://doi.org/10.1145/3352020.3352024

[4] Pooya Davoodi, Chul Gwon, Guangda Lai, and Trevor Morris. 2019. TensorRT inference With TensorFlow. GPU Technology Conference.

[5] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. 2023. Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems* 38 (2023), 100857. https://doi.org/10.1016/j.suscom.2023.100857

[6] Dominic Divakaruni, Peter Jones, Sudipta Sengupta, Liviu Calin. 2018. Amazon Elastic Inference: Reduce Learning Inference Cost. *AWS re:Invent 2018*.

[7] Yi Ge and Monique Jones. 2018. Inference With Intel. AI DevCon 2018.

[8] Mathilde Guillemot, Catherine Heusele, Rodolphe Korichi, Sylvianne Schnebert, and Liming Chen. 2020. Breaking Batch Normalization for better explainability of Deep Neural Networks through Layer-wise Relevance Propagation. *CoRR* abs/2002.11018 (2020). arXiv:2002.11018 https://arxiv.org/abs/2002.11018

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[10] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2261–2269. https://doi.org/10.1109/CVPR.2017.243

[11] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization* (Virtual Event, Republic of Korea) *(CGO '21)*. IEEE Press, 2–14. https://doi.org/10.1109/CGO51591.2021.9370308

[12] Chris Leary and Todd Wang. 2017. XLA: Tensorflow, Compiled!. In *Tensorflow developer summit*.

[13] Yu Liu, Hantian Zhang, Luyuan Zeng, Wentao Wu, and Ce Zhang. 2018. MLbench: Benchmarking Machine Learning Services against Human Experts. *Proc. VLDB Endow.* 11, 10 (jun 2018), 1220–1232. https://doi.org/10.14778/3231751.3231770

[14] P. Pochelu, S. G. Petiton, and B. Conche. 2021. An efficient and flexible inference system for serving heterogeneous ensembles of deep neural networks. In *2021 IEEE International Conference on Big Data (Big Data)*. IEEE Computer Society, Los Alamitos, CA, USA, 5225–5232. https://doi.org/10.1109/BigData52589.2021.9671725

[15] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2020. MLPerf Inference Benchmark. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 446–459. https://doi.org/10.1109/ISCA45697.2020.00045

[16] Dmytro Dzhulgakov Sarah Bird. 2017. ONNX. In *Workshop NIPS2017*.

[17] Rahul Sharma. 2019. Amazon Elastic Inference: Reduce Deep Learning Inference Cost. GPU Technology Conference.

[18] Fuxun Yu, Di Wang, Longfei Shangguan, Minjia Zhang, Chenchen Liu, and Xiang Chen. 2022. A Survey of Multi-Tenant Deep Learning Inference on GPU. *arXiv e-prints*, Article arXiv:2203.09040 (March 2022), arXiv:2203.09040 pages. https://doi.org/10.48550/arXiv.2203.09040 arXiv:2203.09040 [cs.DC]

[19] Wei Zhang, Wei Wei, Lingjie Xu, Lingling Jin, and Cheng Li. 2019. AI Matrix: A Deep Learning Benchmark for Alibaba Data Centers. *CoRR* abs/1909.10562 (2019). arXiv:1909.10562 http://arxiv.org/abs/1909.10562

[20] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. 2016. Accelerating Very Deep Convolutional Networks for Classification and Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 38, 10 (Oct. 2016), 1943–1955. https://doi.org/10.1109/TPAMI.2015.2502579

# Matrix Network Analyzer: a New Decomposition Algorithm for Phase-type Queueing Networks (Work in Progress Paper)

Zhuoyuan Li
Imperial College London
London, United Kingdom
zhuoyuan.li22@imperial.ac.uk

Giuliano Casale
Imperial College London
London, United Kingdom
g.casale@imperial.ac.uk

## ABSTRACT

This paper proposes a new traffic decomposition method called MNA to solve multi-class queueing networks with first-come first-serve stations having phase-type (PH) service, which generalizes the classic QNA method by Whitt. MNA not only supports open queueing networks but also closed networks, which are useful to model concurrency limits in software systems. Using validation models, we show that under low SCV of service time and inter-arrival time, the new method is on average more accurate than QNA. Therefore, MNA can provide better software performance prediction for quality-of-service management tasks.

## CCS CONCEPTS

• **Theory of computation** → *Theory and algorithms for application domains*; *Design and analysis of algorithms*; • **Mathematics of computing** → **Markov processes**; • **Computing methodologies** → **Modeling methodologies**; • **Software and its engineering**;

## KEYWORDS

Queueing Theory; Matrix Analytic Method; Marked Markovian Arrival Process; Phase-type Distribution

**ACM Reference Format:**
Zhuoyuan Li and Giuliano Casale. 2024. Matrix Network Analyzer: a New Decomposition Algorithm for Phase-type Queueing Networks (Work in Progress Paper). In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3629527.3651431

## 1 INTRODUCTION

In software engineering, the stochastic model is a useful tool for analyzing the uncertainty and variability of application workloads. These models offer a way to capture randomness in user behaviors, system loads, and operational environments. By integrating stochastic principles, software engineers can perform nuanced analyses, ranging from system performance and reliability assessments to optimal resource allocation and risk management.

In particular, queueing network models are well-suited to the above aims, helping to analyze the stochastic flow of tasks and data within distributed software systems. By simulating the behavior of resources and the interconnections between different services and components, queueing network models enable a granular analysis of system performance metrics such as response time, throughput, and resource utilization. This level of detail is crucial to accurately predict the behavior of the system under varying load conditions, which is often influenced by stochastic user demands. Furthermore, queueing networks provide insights into potential bottlenecks and resource contention issues, guiding engineers toward targeted optimizations and sizing.

Real workloads often deviate from exponential distributions, yet solution methods for *multi-class* non-exponential queueing networks are limited. In particular, existing methods may not effectively model phase-type (PH) inter-arrival and service times, especially under multiple job classes for which first-come first-serve (FCFS) stations typically do not admit a product-form solution. Using PH models in queueing networks can be beneficial, given that recent work has shown that they can closely fit distributions that are often difficult to represent in a Markovian setting, such as distributions with bounded support [10].

The Queueing Network Analyser (QNA) [13] is a well-known method for non-exponential multiclass queueing networks, focusing on open systems. At the same time, closed and mixed queueing networks also have wide applications such as in layered queueing network analysis. Yet, we notice that for closed queueing networks the matrix analytic method (MAM) [12] commonly used for queueing systems with non-exponential arrivals and service is rarely applied, and even for mixed models a complete theory leveraging MAM is missing. Existing works focus on open models and single-class workloads, e.g. [4, 8, 9]. Multiclass interpolation methods exist, such as the hybrid diffusion-G/G/k methods proposed in [3], however, such methods also do not leverage MAM. Given that MAM is one of the most sophisticated and complete approaches to analyzing non-exponential workloads, we believe that further investigation into the potential of these methods in the context of multiclass queueing networks is warranted. Therefore, in this paper, we introduce a new algorithm, called Matrix Network Analyzer (MNA), which replaces the Langenbach-Belz approximation used in QNA with a MAM solution technique for MMAP/PH/1 nodes [7], allowing us to solve multiple classes queueing networks with FCFS

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651431

queues with PH service time. In various test cases, MNA provides a more accurate result than QNA. Moreover, MNA is shown to extend to closed queueing networks.

In all cases, we find that MNA compares favorably to QNA in the open queueing network model under low SCV of service time and interarrival time and also delivers high accuracy in closed networks, where results are instead compared to discrete-event simulation. We find in particular that the majority of the models enjoy under MNA a small relative approximation error within a 5 percent margin.

In this paper, Section 2 reviews related work and background, Section 3 details the new method, Section 4 presents the experimental results, Section 5 presents a real case example and Section 6 gives conclusions.

# 2 BACKGROUND

## 2.1 Queueing Network Models

A single-class open queueing network consists of a set of $M$ service nodes (or queues), indexed by $i = 1, 2, \ldots, M$, where customers or jobs arrive from outside the network, receive service at one or more nodes, and then leave the network. A closed queueing network similarly consists of a set of $M$ interconnected service nodes (or queues), indexed by, but has a fixed number of customers $N$, circulating within the network. Unlike open queueing networks, there is no arrival from or departure to the outside. The following components characterize the behavior of a network:

- *Arrival Process*: arrivals to node $i$ follow a stochastic process, with rate $\lambda_i$.
- *Service Process at Each Node*: the service times at each node $i$ are random variables. The service discipline (e.g., FCFS also affects performance.
- *Routing Matrix*: a routing matrix $P = [p_{ij}]$ of size $M \times M$ defines the probability $p_{ij}$ that a customer, upon completing service at node $i$, will proceed to node $j$. For an open network, there is also a probability $p_{i0}$ for a customer to leave the network after being served at node $i$.

A multi-class queueing network is similar to a single-class queueing network in terms of network elements, but they have $r$ different customers or jobs classes, indexed by $n = 1, 2, \ldots, r$ and different classes of customers or jobs may have different arrival processes, service processes, and routing matrices.

## 2.2 Markovian Arrival Process (MAP)

In a PH distribution, events occur upon transitions into an absorbing state, whereas in a Markovian arrival process (MAP), the events may be generated by any specific transition between states. Such transitions are referred to as *apparent transitions*, while those that do not lead to an arrival event are termed *hidden transitions*.

MAPs may be specified by a matrix pair that is $D_0 = [C_{i,j}]$ and $D_1 = [D_{i,j}]$. $D_0$ contains all the hidden transitions, while $D_1$ is for the apparent transitions. [2]The generator matrix of the underlying continuous-time Markov chain (CTMC) of this MAP is: $Q = D_0 + D_1$.

The steady-state solution $\pi$ for this CTMC can be calculated using the global balance equations. The steady-state event arrival rate $\bar{\lambda}$ is then: $\bar{\lambda} = \pi D_1 \mathbf{1}$, where $\mathbf{1}$ = (1,1,..,1,1)

**Algorithm 1** MMAP superposition algorithm

```
D_0 = A_0 ⊕ B_0
for i = 1; i ≤ m; i++ do
    D_i = A_i ⊕ 0_{b×b}
end for
for i = 1; i ≤ n; i++ do
    D_{i+m} = B_i ⊕ 0_{a×a}
end for
```

## 2.3 Phase-type Renewal Process

A PH renewal process is an arrival process, where the interarrival time is an independent and identical PH distribution. The renewal process of a PH, of which the initial probability is $\alpha$ and the subgenerator is $T$, can be considered as a MAP, where an arrival event is generated when the embedded Markov chain of this PH transitions into the absorbing state and then immediately transitions to an initial state according to the initial probability $\alpha$. Thus the $D_0$ and $D_1$ of the corresponding MAP is $D_0 = T$ and $D_1 = -T\mathbf{1}\alpha$, where $\mathbf{1}$ is a column vector with the same number of rows with $T$, of which all the element is 1.

## 2.4 Marked Markovian Arrival Process (MMAP)

The marked Markovian Arrival Process is used to model the arrival process of multiple classes where the arrival process of each class is a MAP. An MMAP modeling the arrival process of $n$ classes can be specified by $n + 1$ matrics that is $D_0, D_1, ..., D_{n+1}$. $D_0$ contains all the hidden transitions, and $D_i$ contains the transitions generating a arrival event of class $i$. Using Kronecker sums of the MAP arrival streams that model inter-arrival times of individual classes, an MMAP is readily produced to describe the superposition of the arrival streams [5]; Given two MMAPs modeling the arrival process of $m$ classes and $n$ classes respectively, which can be specified by $A_0, A_1, ..., A_m$ and $B_0, B_1, ..., B_n$, the superposition of these two MMAPs is also an MMAP, which can be used to model the overall arrival process of these $m + n$ classes, specified by $D_0, D_1, ..., D_{m+n}$. Suppose $A_0, A_1, ..., A_m$ are squares matrices of size $a$, and $B_0, B_1, ..., B_n$ are squares matrices of size $b$. The superposition method is shown in Algorithm 1, where $\oplus$ stands for Kronecker sum and $\mathbf{0}_{b\times b}$ stands for a $b \times b$ matrix with all the elements being 0.

# 3 MNA: A NOVEL HYBRID APPROACH

## 3.1 MNA for Open Models

MNA is a novel approach for open queueing networks that integrates QNA, PH fitting, MMAP superposition, and the MAM. This method follows the same network decomposition principle as QNA but relies on the recent developments in MAM to approximate individual nodes. MNA begins by solving first and second-order traffic equations to determine the mean and the Squared Coefficient of Variation (SCV) of the interarrival times at each node. The notations used below are all listed in Table 1. The first-order traffic equations

are:

$$\lambda_{i,r} = \lambda_{0i,r} + \sum_{j=1}^{N} \lambda_{j,r} p_{ji,r}, \tag{1}$$

$$\lambda_i = \sum_{r=1}^{R} \lambda_{i,r}, \tag{2}$$

$$\lambda_{ij,r} = \lambda_{i,r} p_{ij,r}, \tag{3}$$

After solving these equations, one can obtain all the means of inter-arrival times for each class at all nodes. Thus, the utilization can be calculated as:

$$\rho_{i,r} = \frac{\lambda_{i,r}}{m_i \mu_{i,r}}, \tag{4}$$

$$\rho_i = \sum_{r=1}^{R} \rho_{i,r}, \tag{5}$$

$$\mu_i = \frac{\lambda_i}{\rho_i}, \tag{6}$$

Then, the second-order traffic equations are:

$$C_{S_i}^2 = -1 + \sum_{r=1}^{R} \frac{\lambda_{i,r}}{\lambda_i} (\frac{\mu_i}{m_i \mu_{i,r}})^2 (C_{S_{i,r}}^2 + 1) \tag{7}$$

$$C_{A_{i,j}}^2 = \frac{1}{\lambda_{i,j}} \sum_{j=0}^{N} C_{ji,r}^2 \lambda_{j,r} p_{ji,r} \tag{8}$$

$$C_{A_i}^2 = \frac{1}{\lambda_i} \sum_{r=1}^{R} C_{A_{i,j}}^2 \lambda_{i,j} \tag{9}$$

$$C_{D_i}^2 = 1 + \frac{\rho_i^2 (C_{S_i}^2 - 1)}{\sqrt{m_i}} + (1 - \rho_i^2)(C_{A_i}^2 - 1) \tag{10}$$

$$C_{ij,r}^2 = 1 + p_{ij,r} (C_{D_i}^2 - 1) \tag{11}$$

Upon solving the second-order traffic equations, the means and SCVs of inter-arrival times for each class at all nodes are obtained. The above equations are from QNA for the multi-class open model. [6]

Subsequently, each node is addressed individually. QNA solves the network node-by-node by using the Langenbach-Belz approximation:

$$\alpha_{m_i} = \begin{cases} \frac{\rho_i^{m_i} + \rho_i}{2}, & \text{if } \rho_i > 0.7, \\ \rho_i^{\frac{m_i+1}{2}}, & \text{if } \rho_i < 0.7, \end{cases} \tag{12}$$

$$W_{iq} \approx \frac{\alpha_{m_i}}{\bar{\mu_i}} \frac{1}{1 - \rho_i} \frac{C_{A_i}^2 + C_{S_i}^2}{2}, \tag{13}$$

$$L_{i,r} = \frac{\lambda_{i,r}}{\mu_{i,r}} + \lambda_{i,r} W_{iq}. \tag{14}$$

Instead, for each node, a three-moment matching algorithm [1] is first applied to fit the mean and SCV of interarrival times, and then use the corresponding PH renewal process as a MAP of this class to this node. Next, we superpose these MAPs to produce an MMAP as the overall arrival process to this node. Then, every single queue in the network is solved as a MMAP[R]/PH[R]/1/FCFS queue, allowing to account for class arrival cross-correlations. Indeed, while the superposition of Poisson processes is Poisson, the superposition even of renewal (i.i.d.) processes can produce non-renewal processes (non-i.i.d.) [11], thus the MMAP representation helps us to capture interarrival time covariances introduces in this fashion. He [7] proposes a matrix analytic method for calculating the margin distributions of the queue length of different classes in an MMAP[R]/PH[R]/1/FCFS queue. Finally, the mean queue length of each class can be calculated according to its margin distribution. Combining the utilization acquired in the previous traffic equation, other metrics of this node can be calculated.

**Table 1: Notaitons and descriptions of MNA**

| Notation | Description |
|---|---|
| $\lambda_{ij,r}$ | Mean arrival rate of class $r$ customers from node $i$ to node $j$ |
| $\lambda_{i,r}$ | Mean arrival (departure) rate of class $r$ customers to (from) node $i$ |
| $\lambda_i$ | Mean aggregate arrival (departure) rate to (from) node $j$ |
| $\mu_{i,r}$ | Mean service rate of class $r$ customers at node $i$ |
| $\mu_i$ | Mean aggregate service rate at node $i$ |
| $\rho_{i,j}$ | Utilization of node $i$ due to customers of class $r$ |
| $\rho_i$ | Utilization of node $i$ |
| $m_i$ | Numbers of servers of node $i$ |
| $C_{ij,r}^2$ | SCV of time between two consecutive class $r$ customers going from node $i$ to node $j$ |
| $C_{Ai,r}^2$ | SCV of interarrival time for class $r$ to node $i$ |
| $C_{Ai}^2$ | Aggregate SCV of interarrival time to node $i$ |
| $C_{Di}^2$ | Aggregate SCV of node $i$ inter-departure times |
| $C_{S_i}^2$ | Aggregate SCV of service time of node i |
| $C_{S_{i,r}}^2$ | SCV of service time for customer class r at node i |
| $p_{ij,r}$ | Routing probability of Class r from node $i$ to $j$ |
| $N$ | Numbers of queueing nodes |
| $R$ | Numbers of classes |
| $Q_{i,r}$ | Queue length of customer class r at node i |
| $n_r$ | Numbers of jobs for class r. |

## 3.2 MNA for Closed Models

The MNA method for closed queueing networks is shown in Algorithm 2.

*Initialization (Lines 1-4):* $\lambda_{i,r}$ is the arrival rate of class $r$ to node $i$, while $\mu_{i,r}$ is the service rate of class $r$ to node $i$. These lines set the upper bound of the arrival rate of class $r$ to the reference node equal to the bottleneck service rate of this class.

*Initial Guess (Lines 5-7):* $\lambda$ is a vector contains the guessed arrival rates of all the job classes to the reference node, namely $\lambda = (\lambda_{1,1}, \lambda_{1,2}, ...)$. In this first iteration, an initial guess is made that the arrival rate of class $r$ to the reference node is equal to the bottleneck service rate of this class.

**Algorithm 2** MNA for closed queueing networks

1: **for** $r = 1; r \leq R; r++$ **do**
2: $\lambda_{1,r}^{\mathrm{UB}} = \min \mu_{i,r}$
3: $\lambda_{1,r}^{\mathrm{LB}} = 0$
4: **end for**
5: **for** $it = 1; it <$ it_max; $it++$ **do**
6: **if** $it == 1$ **then**
7: $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\mathrm{UB}}$
8: **else**
9: **if** $(\max_i(QN_r - n_r) <$ threshold$)$ **then**
10: **break**
11: **else**
12: $r^* = \arg\max_r \left(\frac{QN_r - n_r}{QN_r}\right)$
13: **if** $(QN_{r^*} > n_{r^*})$ **then**
14: $\lambda_{1,r^*}^{\mathrm{UB}} = \lambda_{1,r^*}$
15: **else**
16: $\lambda_{1,r^*}^{\mathrm{LB}} = \lambda_{1,r^*}$
17: **end if**
18: **end if**
19: $\boldsymbol{\lambda} = (\boldsymbol{\lambda}^{\mathrm{UB}} + \boldsymbol{\lambda}^{\mathrm{LB}})/2$
20: **end if**
21: Solve the network under arrival rate of $\lambda$, using MNA algorithm for open queueing networks
22: $QN_r = \sum_{i=1}^{N} Q_{i,r}$
23: **end for**

*Arrival Rate Adjustment (Lines 8-20):* This step are performing a fixed point iteration, making the guessed arrival rate converge to the real arrival rate. $QN_r$ is the sum of the queueing length of class $r$, and $n_r$ is the population of job class $r$ in this network. The iteration ends if the difference between $QN_r$ and $n_r$ is small enough for every job class $r$. Otherwise, MNA adjusts the arrival rate of the class, which has the largest population relative error. If $QN_r$ is larger than $n_r$, then we set the upper bound of the arrival rate of this class to the guessed value, otherwise, we set the lower bound to the guessed value. MNA uses the average value of the upper bound and lower bound as the new guessed value in the next iteration.

*Network Analysis (Line 21-22):* Using the guessed arrival rate, the closed queueing network can be solved as an open queueing network. In each iteration, a method very similar to MNA for the open queueing network is called to evaluate the current state of the network given the current arrival rates $\boldsymbol{\lambda}$. The only difference is that, when solving closed models, after getting the margin distribution of the queue length of class $r$ in a node, instead of directly using the mean of this distribution, the mean queue length of this class is calculated by $\sum_{i=1}^{n_r} \frac{P(x=i)i}{\sum_{k=1}^{n_r} P(x=k)}$, where $P(x=i)$ is the probability of this node having a queue length of $i$. After achieving the queue length of class $r$ at each node, the total population of this class, namely $QN_r$, can be calculated by the sum of queue lengths at all the nodes. The outcome of this analysis feeds into the next iteration for further adjustment.

In summary, this algorithm initially guesses the arrival rate of the reference node, transforming the closed queueing network into an open queueing network, then applies a fixed point iteration, which adapts the rates based on the network performance in each iteration, aiming to achieve a balance between the actual and desired populations in each class.

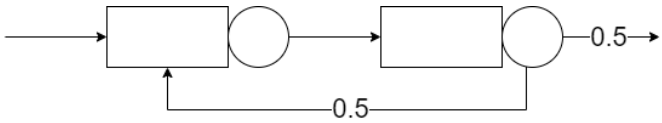



**Figure 1: A feedback loop open queueing network**

**Table 2: Performance of MNA for single-class open queueing network**

| node | mean ARE | models within 5% ARE |
|---|---|---|
| queue 1 | 0.0201 | 951/1000 |
| queue 2 | 0.0242 | 908/1000 |
| overall | 0.0221 | 871/1000 |

## 4 NUMERICAL EVALUATION OF MNA

In this chapter, the improvements of MNA will be shown through a selection of experimental cases, highlighting its higher accuracy compared to QNA. These experiments are conducted with assistance from LINE, a Matlab toolbox created by the QORE lab at Imperial College London. LINE is designed for resolving complex queueing network models using various algorithms or simulations. [3]

### 4.1 Model Design

To facilitate a comparative evaluation of the performance between QNA and MNA, we have designed a benchmark using an open feedback queuing network with a feedback loop. As depicted in Figure 1, this network comprises several key elements: a source with an arrival rate $\lambda$, and two FCFS queues with individual service rates $\mu_1$ and $\mu_2$, respectively.

Jobs generated by the source initially enter Queue 1. After being serviced in Queue 1, they are transferred to Queue 2. Upon completion of service in Queue 2, each job has a probability of 0.5 of proceeding directly to a sink. Conversely, there is a probability 0.5 that the job will be re-routed back to Queue 1.

### 4.2 Single-class Open Queueing Networks

Consider the following example and its numerical result in Table 2 & Figure 2: the interarrival time, and the service time of node 1 and node 2 are all PH distributed. the mean interarrival time is generated by a random variable uniformly distributed between 40 and 45, the mean service time of node 1 is generated by a uniform distribution between 5 and 8, and the mean service time of node 2 is generated by a uniform distribution between 3 and 6. The SCVs for interarrival time and service time of node 1 and node 2 are all generated by a uniform distribution a uniform distribution between 0.01 and 0.8. Through the random selection of 1000 instances, these conditions are examined empirically.

In the experimental analysis, it is observed in Table 2 that 90 percent of the results obtained from MNA exhibited an Absolute

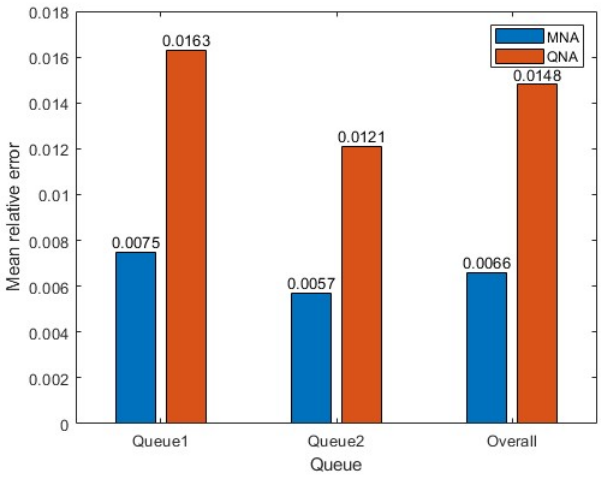



**Figure 2: MAE for single-class open queueing networks**

**Table 3: Interarrival time parameters for multi-class open queueing network test**

| class | mean | SCV |
|---|---|---|
| 1 | $U(40, 50)$ | 1/4 |
| 2 | $U(52, 57)$ | 1/5 |

**Table 4: Service time parameter for multi-class open queueing network test**

| node | class | mean | SCV |
|---|---|---|---|
| 1 | 1 | $U(2, 5)$ | 1/6 |
| 1 | 2 | $U(2, 5)$ | 1/7 |
| 2 | 1 | $U(3, 6)$ | 1/9 |
| 2 | 2 | $U(3, 5)$ | 1/2 |

Relative Error (ARE) within the 5 percent threshold. Additionally, MNA provides a better result for both node 1 and node 2 than QNA. This is evidenced by the lower Mean Absolute Errors (MAE) observed for MNA in Figure 2.

## 4.3 Multiclass Open Queueing Networks

Considering the following example which uses the same network routing as the last example but has multiple job classes. The interarrival time, and the service time are all phase-type distributed. The means and SCVs are generated as shown in the table 3 and 4 where all the means follow uniform distributions and SCVs are fixed values. For this example, without loss of generality, we set SCV as a fixed value. This is because randomly generated values may include some irrational numbers, which may lead to a MAP with extremely large state space which increases the execution time.

As is shown in Table 5 and Figure 3, MNA demonstrates high accuracy for multi-class scenarios. Approximately 95% of the samples exhibit an ARE within a 5% margin. When compared to QNA,

**Table 5: Performance of MNA for multi-class queueing network**

| node and class | mean ARE | models within 5% ARE |
|---|---|---|
| queue 1, class 1 | 0.0115 | 993/1000 |
| queue 2, class 1 | 0.0220 | 932/1000 |
| queue 1, class 2 | 0.0201 | 985/1000 |
| queue 2, class 2 | 0.0256 | 891/1000 |
| overall | 0.0198 | 947/1000 |

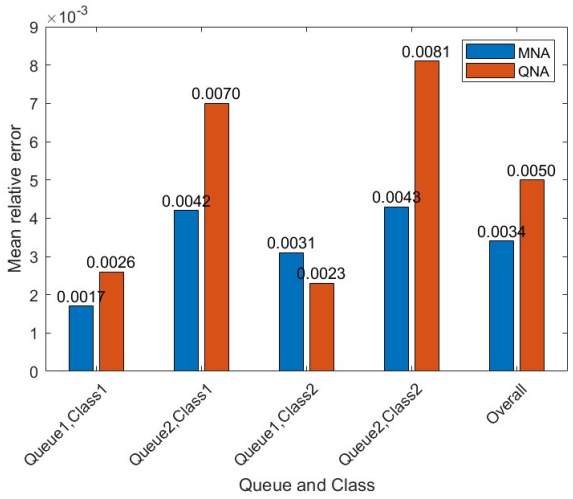



**Figure 3: MAE for multi-class open queueing network**

**Table 6: Performance of MNA for single-class closed models**

| node | Mean ARE | models within 5% ARE |
|---|---|---|
| queue 1 | 0.0247 | 942/1000 |
| queue 2 | 0.0161 | 967/1000 |
| overall | 0.0204 | 951/1000 |

MNA offers more accurate results. While there are instances where MNA does not perform as well as QNA for specific classes at certain nodes, it consistently outperforms QNA for the whole system.

## 4.4 Closed Queueing Networks

Consider this example: a closed queueing network with 3 nodes and 1 class. The first node is a delay node, and the other two are FCFS queueing nodes. The interarrival time, and the service time are all phase-type distributed. The service time of the delay node has a mean of $\mu$, where $\mu$ follows a uniform distribution $U(1, 3)$, and the SCV is 4. The service time of the first FCFS node has a mean of $\mu$, where $\mu$ follows a uniform distribution $U(2, 4)$, and the SCV is 4. The service time of the second FCFS node has a mean of $\mu$, where $\mu$ follows a uniform distribution $U(1, 4)$, and the SCV is 4. The number of jobs is 4, and the jobs follow a circular routing, namely from the delay node to the first FCFS node, then to the Second FCFS node, and finally back to the delay node.

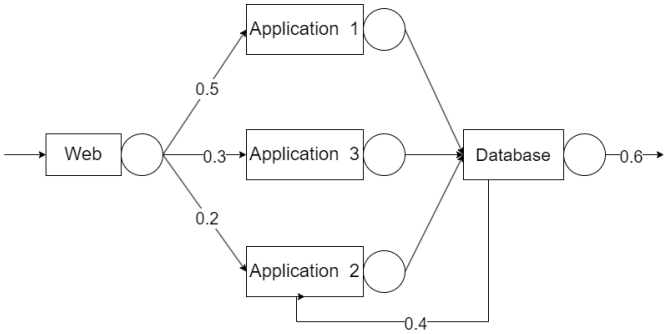



**Figure 4: Routing for class 1**

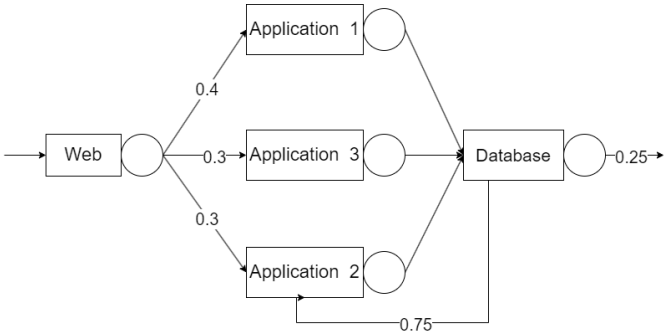



**Figure 5: Routing for class 2**

As is shown in Table 6, there is only a 2 percent mean relative error between the result of MNA and simulation. 95% of the samples exhibit an ARE within a 5% margin.

## 5 APPLICATION AND EXAMPLE

We model as a queueing network a three-tier e-commerce system consisting of the following components: *Web Server*: the first point of contact is a web server that handles initial customer requests. *Application Servers*: once the web server processes the initial request, it forwards the customer to one of the three available application servers, depending on their specific needs. *Database Server*: each application server is connected to a central database server responsible for data access.

The system serves two different classes of customers. Upon visiting the website, customers first arrive at the web server. The web server evaluates their needs and routes them to the most appropriate application server for further processing. Once the application server completes its tasks, it interacts with the database server for data storage or retrieval. After being served by the database server, customers have two options: they may continue using the website, in which case they are sent back to the application server for additional services. Alternatively, they may choose to leave the system.

The routing probability of class 1 and class 2 are shown in Figure 4 and Figure 5 respectively. The interarrival time, and the service time are all phase-type distributed. The means of interarrival time of class 1 and class 2 are 4 and 5 respectively, and the SCVs of interarrival time are all 1/10. The means and SCVs of the service time are generated as shown in Table 7. For this example, QNA provides a result with a relative error of 8.6 percent while MNA provides a more accurate result with a relative error of 1.5 percent.

**Table 7: Service time parameters for real case example**

| node | classes | mean | SCV |
|---|---|---|---|
| Application 1 | 1,2 | 0.5 | 1/10 |
| Application 2 | 1 | 1 | 1/10 |
| Application 2 | 2 | 0.3 | 1/10 |
| Application 3 | 1, 2 | 0.5 | 1/10 |
| Database | 1 | 0.3 | 1/10 |
| Database | 2 | 0.5 | 1/10 |

## 6 CONCLUSION

In this paper, we have proposed MNA, a new algorithm for solving multiclass PH queueing networks that leverages the matrix-analytic method. The method has been shown to improve the accuracy of the class QNA algorithm.

In future work, we seek to integrate additional features into MNA. In particular, the method should be extended to incorporate functionalities such as self-loops, class-switching, and mixed workloads. A comparison with gradient-based methods to seek the fixed point would also be beneficial.

Additionally, throughput calculation for closed multi-class queueing networks may face challenges due to its use of fixed iterations, which may not converge. A more efficient and accurate method, perhaps based on gradient search, may be needed.

## REFERENCES

[1] Andrea Bobbio, Andras Horvath, and M. Telek. 2005. Matching Three Moments with Minimal Acyclic Phase Type Distributions. *Stochastic Models* 21 (01 2005), 303–326. https://doi.org/10.1081/STM-200056210
[2] Peter Buchholz, Jan Kriege, and Iryna Felko. 2014. *Input modeling with phase-type distributions and Markov models: theory and applications.* Springer.
[3] Giuliano Casale. 2021. Integrated performance evaluation of extended queueing network models with line. In *Proceedings of the Winter Simulation Conference* (Orlando, Florida) *(WSC '20)*. IEEE Press, 2377–2388.
[4] Giuliano Casale and Peter Harrison. 2012. A class of tractable models for runtime performance evaluation. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. 63–74.
[5] Giuliano Casale, Andrea Sansottera, and Paolo Cremonesi. 2016. Compact Markov-modulated models for multiclass trace fitting. *European Journal of Operational Research* 255, 3 (2016), 822–833. https://doi.org/10.1016/j.ejor.2016.06.005
[6] Natarajan Gautam. 2012. Analysis of queues: methods and applications. CRC press.
[7] Qiming He. 2012. Analysis of a continuous time SM [K]/PH [K]/1/FCFS queue: Age process, sojourn times, and queue lengths. *Journal of Systems Science and Complexity* 25 (2012), 133–155.
[8] Armin Heindl. 2001. Decomposition of general tandem queueing networks with MMPP input. *Performance Evaluation* 44, 1-4 (2001), 5–23.
[9] András Horváth, Gábor Horváth, and Miklós Telek. 2010. A joint moments based analysis of networks of MAP/MAP/1 queues. *Performance Evaluation* 67, 9 (2010), 759–778.
[10] A Horváth and E Vicario. 2023. Construction of phase type distributions by Bernstein exponentials. In *European Workshop on Performance Engineering*. Springer, 201–215.
[11] MF Neuts. 1989. Structured Stochastic Matrices of M/G/1 Type and Their Applications. *Marcel Dekker* (1989).
[12] Marcel F Neuts. 1994. *Matrix-geometric solutions in stochastic models: an algorithmic approach.* Courier Corporation.
[13] Ward Whitt. 1983. *The queueing network analyzer. The bell system technical journal* 62, 9 (1983), 2779–2815.

# Towards Efficient Diagnosis of Performance Bottlenecks in Microservice-Based Applications (Work In Progress paper)

Adel Belkhiri
École Polytechnique de Montréal
Montreal, Canada
adel.belkhiri@polymtl.ca

Maroua Ben Attia
Humanitas Solutions
Montreal, Canada
maroua@humanitas.io

Felipe Gohring de Magalhaes
École Polytechnique de Montréal
Montreal, Canada
felipe.gohring-de-magalhaes@polymtl.ca

Gabriela Nicolescu
École Polytechnique de Montréal
Montreal, Canada
gabriela.nicolescu@polymtl.ca

## ABSTRACT

Microservices have been a cornerstone for building scalable, flexible, and robust applications, thereby enabling service providers to enhance their systems' resilience and fault tolerance. However, adopting this architecture has often led to many challenges, particularly when pinpointing performance bottlenecks and diagnosing their underlying causes. Various tools have been developed to bridge this gap and facilitate comprehensive observability in microservice ecosystems. While these tools are effective at detecting latency-related anomalies, they often fall short of isolating the root causes of these problems. In this paper, we present a novel method for identifying and analyzing performance anomalies in microservice-based applications by leveraging cross-layer tracing techniques. Our method uniquely integrates system resource metrics—such as CPU, disk, and network consumption—with each user request, providing a multi-dimensional view for diagnosing performance issues. Through the use of sequential pattern mining, this method effectively isolates aberrant execution behaviors and helps identify their root causes. Our experimental evaluations demonstrate its efficiency in diagnosing a wide range of performance anomalies.

## CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

## KEYWORDS

Microservices, Performance analysis, Software tracing, Distributed systems

**ACM Reference Format:**
Adel Belkhiri, Maroua Ben Attia, Felipe Gohring de Magalhaes, and Gabriela Nicolescu. 2024. Towards Efficient Diagnosis of Performance Bottlenecks in Microservice-Based Applications (Work In Progress paper). In *Companion of the 15th ACM/SPEC Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651432

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom.*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651432

## 1 INTRODUCTION

Microservices have emerged as a paradigm of choice for cloud-based applications, thanks to their scalability and flexibility. Unlike the monolithic architecture which encompasses all functionalities within a single codebase, microservices break down applications into a set of autonomous, self-contained, and single-purpose services. These services operate independently and communicate via well-defined interfaces and lightweight APIs (e.g., RESTful APIs). Such modularity enables agile scaling and promotes polyglot programming, allowing services to be developed in the languages and frameworks best suited for their tasks. Nevertheless, the compartmentalization of services often introduces challenges, particularly in debugging performance bottlenecks. In a microservice environment, the processing of user requests often requires coordinated actions from multiple services. These intricate interdependencies among microservices create a complex chain of dependencies, where a performance bottleneck in one service can trigger cascading effects that compromise the efficiency of the entire application. Identifying the culprit service and isolating the root causes of performance degradation within such a decentralized architecture proves to be a complex endeavor.

Distributed tracing [11, 15, 17, 25, 29] is a powerful method for monitoring user requests as they move through the various components of a distributed application. It tracks the end-to-end execution of user requests through the insertion of unique identifiers into requests and the propagation of metadata between processes and system components. Hence, this method provides a comprehensive view of each request's end-to-end execution, shedding light on its life cycle from initiation to completion. A "trace" represents the journey of a single request, documenting the sequence of operations it undergoes [4]. Within a trace, individual work units are captured as "spans," each corresponding to an action (e.g., a function call, database query, or instruction blocks) executed by a service or component. Spans are nested within traces to illustrate the hierarchical relationships between different operations, offering a detailed and structured view of how a request is processed across multiple services. While distributed tracing effectively captures the flow and timing of requests, it falls short of pinpointing the root causes of unexpected latencies. The reason is that it only collects

high-level information. This limitation is especially pronounced when unexpected delays stem from operating system-level resource contention, such as waiting for CPU, disk, network, or lock availability.

To address this limitation, some efforts have attempted to enrich traces generated through distributed tracing with application- and kernel-level logs, aiming to identify slow code paths, contention for resources, and load imbalances [3, 26]. However, these approaches often fail at detecting and diagnosing transient and short-lived performance problems. Another approach involves leveraging vertical context propagation to inject application-level events into kernel traces [2, 4]. This provides a granular understanding of system behavior but comes at the cost of additional complexity and significant overhead. Additionally, several statistical and machine-learning methods have been explored for analyzing the performance of distributed applications [5, 10, 16, 18, 21, 24, 28]. While these methods offer powerful analytical capabilities, they come with many limitations, such as low detection accuracy and computational inefficiency. In short, although the proposed approaches offer insights into different types of performance problems, they mostly struggle to accurately identify the root causes of these issues.

In this paper, we propose a novel approach for identifying performance problems in microservice applications and uncovering their underlying causes. Based on this approach, we implement a cross-layer analysis enabling the characterization of request executions. Additionally, we leverage a combination of distributed and software tracing techniques to capture both kernel- and application-level events. We use a small subset of kernel events to conduct fine-grained critical path analysis of service threads, and application-level events to delimit the spans of operations involved in request processing. By utilizing a sequential pattern mining technique, we extract sequences of thread states that characterize the behavior within each request category. Using these normative patterns as a basis, we identify anomalous request executions. Anomalies are flagged when the observed behavior diverges significantly from the established patterns.

The rest of the paper is organized as follows. Section 2 introduces our approach and elaborates on the design details of the implemented framework. Section 3 evaluates the effectiveness of our framework in practical scenarios through an illustrative use case. It also assesses the overhead it induces and discusses avenues for potential improvements. Section 4 reviews relevant works in the field that have informed our research. Finally, Section 5 concludes this paper and outlines directions for future work.

## 2 PROPOSED SOLUTION

The framework we developed to implement this approach is specifically designed for seamless integration with distributed tracing infrastructures supporting OpenTelemetry (OTel) [6]. OTel is an open-source initiative that provides a comprehensive suite of APIs, libraries, agents, and instrumentation designed to enhance observability in distributed applications. Its main goal is to provide developers with a unified way to collect distributed traces and metrics through instrumentation. The vendor-neutral design of OTel makes it compatible with a wide range of distributed tracers, including but not limited to Jaeger [15] and Zipkin [29]. Consequently, this design consideration greatly simplifies the integration of our framework into existing systems.

Our framework aims to detect performance issues in microservice-based applications and uncover their root causes through offline analysis. To pinpoint performance anomalies, our framework requires two separate sets of trace data - referred to as the 'baseline' and 'test' datasets. The baseline dataset is used to generate a basis for modeling the software's normal behavior, while the test dataset is evaluated against this baseline to identify any deviation or abnormality. Fig. 1 depicts the architecture of our framework and outlines its key operations. These elements will be discussed in greater details in the following sections.

### 2.1 Capturing Execution Traces

Our approach uses cross-layer tracing to collect fine-grained data that characterizes resource consumption per request. Therefore, to avoid the overhead associated with vertical context propagation and the need to modify the application source code, we chose to instrument OTel libraries using the Linux Trace Toolkit Next Generation (LTTng) [7]. LTTng is a high-throughput tracer for Linux-based systems that is designed for low-overhead tracing of applications at kernel and user-space levels. The instrumentation we added to OTel aims to emit userspace events each time a service starts or finishes the processing of a request. Hence, we inserted tracepoints into the API methods responsible for starting and ending spans. For example, to gather data from C++-based microservices, we instrumented the *Tracer* class's *StartSpan()* and *end()* methods within the opentelemetry-cpp library. We extended our instrumentation to multiple OTel libraries, as a microservice application may consist of services developed in various programming languages. Our analysis requires also gathering specific kernel events to construct critical paths for request executions. The Linux kernel comes with hundreds of tracepoints, allowing us to capture needed events without additional instrumentation. A description of a subset of leveraged kernel events is provided in Table 1.

### 2.2 Classification of Traces

After collecting execution traces, our framework starts analyzing them to identify potential performance anomalies. The analysis is based on the hypothesis that operations in traces of the same type should exhibit similar performance characteristics when processed by the same services. Therefore, categorizing traces based on their types is critical. We consider traces to be of the same type if they present the same workflow. A trace workflow outlines the order in which operations and requests are executed within individual processes and across the various services that compose a microservice-based application. Our framework recreates trace workflows by generating tree structures from the operation names exported via the userspace events mentioned earlier. The tree's root node is labeled with the name of the initiating request, which is also known as the root span. The remaining nodes are labeled with the names of their respective requests/operations. Unique identifiers for tasks performed by services are formulated by pairing the "service_name" with the "operation_name". After building the workflow trees, we use a hash function to generate an identifier for the trace type that captures both the labels of the nodes and their

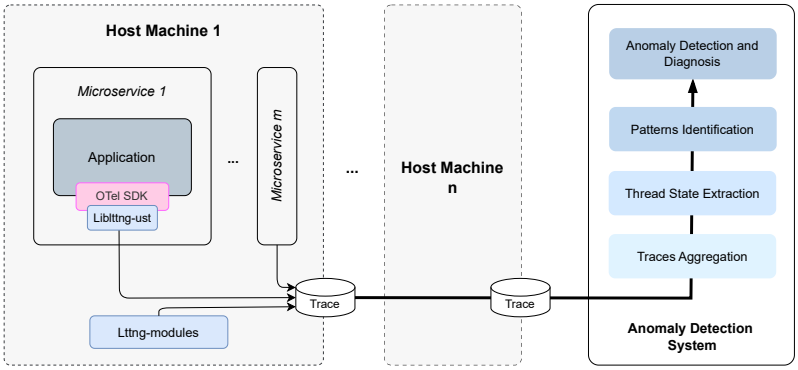



**Figure 1: The operation of our framework is based on collecting kernel-level events and leveraging an instrumented version of OTel to add information about the start and end of spans to the kernel trace**

**Table 1: A subset of the kernel events required for our analysis**

| Tracepoint | Description |
|---|---|
| *sched_switch* | Signals that a new thread has taken over from a previously active thread on a CPU. |
| *sched_wakeup* | Triggered when a thread, previously in a blocked state, is now ready to execute. |
| *softirq_entry/exit* | Indicates the start/end of a software interrupt handler's execution. |
| *irq_handler_entry/exit* | Indicates the start/end of a hardware interrupt handler's execution. |
| *timer_expire_entry/exit* | Indicates the start/end of a timer interrupt's execution. |
| *sched_process_fork* | Fires when a new process is created by the kernel. |

relative positions within the tree structure. Hence, traces whose workflows produce identical type identifiers are classified in the same category.

## 2.3 Extraction of Thread States

Our second hypothesis is that when processing operations of the same type, the service threads will follow a consistent sequence of states. For example, let us consider a basic authentication service and its thread states during operation. When a login request is received, the service initially validates the provided username and password against predefined criteria (state: running). It then retrieves the associated hashed password from a disk (or a database), keyed by the username (state: *blocked for disk*). After that, the service compares the stored hashed password with the hashed version of the received password (state: *running*). Finally, the outcome of this comparison is transmitted back to the originating service (state: *blocked for network*). Therefore, it is reasonable to assume that the service thread will follow the same sequence of states when processing future requests. If it deviates from the expected sequence, either there is something wrong with its behavior or other unknown factors at play. This unexpected behavior could potentially reflect performance issues like contention for resources, defective hardware, or slow functions.

To ascertain the states through which operations progress during their execution, our framework identifies the critical path of the subsequent service threads. In software engineering, the "critical path" refers to the sequence of dependencies that inherently limit the speed at which a thread can be completed [27]. Threads rely on hardware and software resources for execution. Their hardware dependencies include but are not limited to the need for CPU time, disk access, or network bandwidth. As for the software dependencies, it may involve waiting for data from other threads or requiring certain locks or semaphores to be available for synchronization. Practitioners often use critical path analysis to assess software resource bottlenecks and gain insight into a process's interactions with system resources and other processes.

Based on the algorithm proposed in [12], we developed an analysis in Trace Compass [9] to extract the critical path of services' threads and obtain their states during operations' execution. We converted operations execution into a text-based representation, wherein each unique thread state is encoded as a distinct letter of the alphabet. For example, the "Running" state is represented by the letter 'R', the state "blocked for Disk" is represented by 'D', the state "blocked for Network" is denoted by 'N', and the state "blocked for Timer" is denoted by 'T'. It is worth noting that our analysis is based on 8 thread states as we excluded some states that are not indicative of application behavior (e.g., the 'Interrupted' and 'Preempted' states). In addition, we introduced an extra state, symbolized by the letter 'Z', to represent the execution of sub-operations. Our framework leverages this state to achieve accurate anomaly detection as the conducted analysis is guided by the operations hierarchy (see Fig. 2).

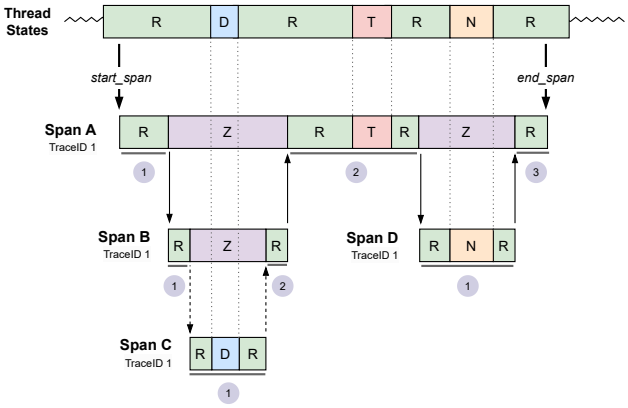



**Figure 2: Thread states are transposed to corresponding spans and sub-spans. Spans' intervals are highlighted with a bold grey line underneath it**

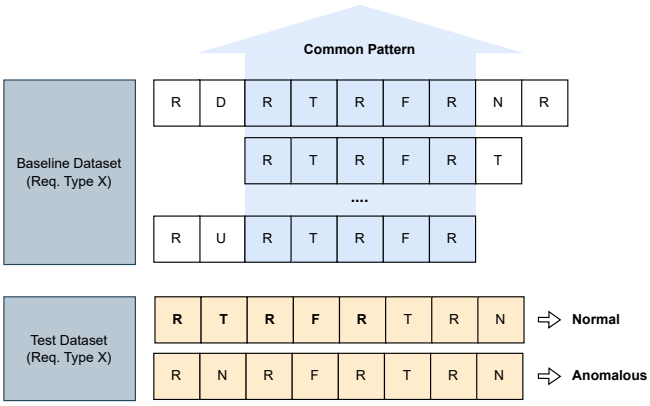



**Figure 3: Sequential pattern mining is used to extract execution state patterns from same-type trace operations.**

## 2.4 Frequent Pattern Mining

Establishing a baseline for what is considered normal behavior while processing operations is crucial for detecting performance bottlenecks and anomalies. Our approach focuses on identifying recurring patterns in thread execution states during these periods. We achieve this through sequential pattern mining, a technique specifically designed to uncover recurring subsequences within a set of sequences. As we can observe in Fig. 3, these subsequences correspond to frequent sequences of execution states. We assess the relevance of discovered state subsequences based on their lengths and frequency of occurrence.

Sequential pattern mining is not limited to our context but finds broad applications in various areas like financial market prediction, and text analysis. It is widely used to identify frequently occurring ordered events or subsequences in various types of datasets. Formally speaking, let $D = \{s_1, s_2, \ldots, s_n\}$ be a set of sequences, where each sequence $s_i$ is an ordered list of items $\langle a_1, a_2, \ldots, a_m \rangle$. An itemset $X$ is said to be a "sequential pattern" if it appears in at least *minsup* number of sequences in $D$, where *minsup* is a predefined minimum support threshold. On the other hand, a sequence $s = \langle a_1, a_2, \ldots, a_m \rangle$ is said to "contain" an itemset $X = \{x_1, x_2, \ldots, x_k\}$ if there exists a subsequence $\langle a_{i_1}, a_{i_2}, \ldots, a_{i_k} \rangle$ such that $a_{i_j} = x_j$ for all $j$ from 1 to $k$. The aim is to find all such itemsets $X$ that satisfy the minimum support condition in the given dataset $D$.

Our approach is based on the implementation of the algorithm proposed in [20] to identify all closed sequential patterns of thread states that are present in the dataset. A closed sequential pattern can be defined as a sequential pattern that is not a strict subset of any other pattern with identical support. This algorithm further allows us to impose constraints on the gaps between states, thereby offering a more flexible way to mine recurring sequential states. The minimal support value required for identifying patterns is a user-defined parameter, but we recommend setting it at 95% or higher. Setting this parameter at a high value allows prioritizing patterns that are more common in the dataset and filtering out infrequent ones. This would improve the effectiveness of our anomaly detection analysis and ensure its scalability for larger datasets.

## 2.5 Performance Anomaly Diagnosis

Our methodology, as we explained in the previous section, relies on extracting from the baseline dataset patterns that encode the service runtime behavior during operations execution. This involves clustering similar traces based on their type identifiers. Then, critical path analysis is used to extract, for each operation, threads' execution states as illustrated in Fig. 2. Moreover, the execution of each operation is divided into intervals based on the occurrence of the 'Z' state. Thus, the number of intervals is equal to the number of sub-operations plus one. For instance, in Fig. 2, *Span A* is divided into three intervals, *Span B* into two, and *Span C* and *Span D* each into a single interval. Subsequently, we apply sequential pattern mining to identify patterns in thread states within these intervals, and we compute the minimum and maximum latencies for each state encompassed by these patterns. These latency thresholds are established using the three-sigma rule, thereby enriching the pattern states with mean and standard deviation information.

The second step in our methodology is the evaluation of the test dataset to ascertain whether its operations are normal or anomalous. For each trace in this set, we identify its type and extract the thread states occurring while executing its operations. We also determine the states that correspond to the operation intervals and require validation. By comparing the generated trace type identifiers with type identifiers in the baseline dataset, we determine the frequent state patterns that must be validated against the state sequences in each operation interval. Therefore, for every operation interval, we check if the observed state sequence matches the expected pattern. If this is the case, we check whether the states of the sequence matching the pattern are within the identified limits. This is done through a bottom-up approach, where sub-spans are evaluated before their parent spans. If a discrepancy is found during this verification process, the operation in question is flagged as anomalous, triggering a more detailed investigation to pinpoint the cause of the deviation. Furthermore, this hierarchical evaluation serves as a structured way to address potential issues at the granular level of sub-spans and execution intervals, which simplifies isolating and resolving performance problems.

# 3 EVALUATION AND DISCUSSION

To demonstrate the effectiveness of our approach in diagnosing performance anomalies in microservice applications, we leverage "Bank-of-Sirius," an HTTP-enabled web application that emulates a banking system [23]. This application allows users to create bank accounts and execute financial transactions. We chose Bank-of-Sirius for our evaluation because it comes pre-instrumented with OTel and features a diverse architecture, comprising nine microservices implemented in Python, Java, and C (Fig. 4).

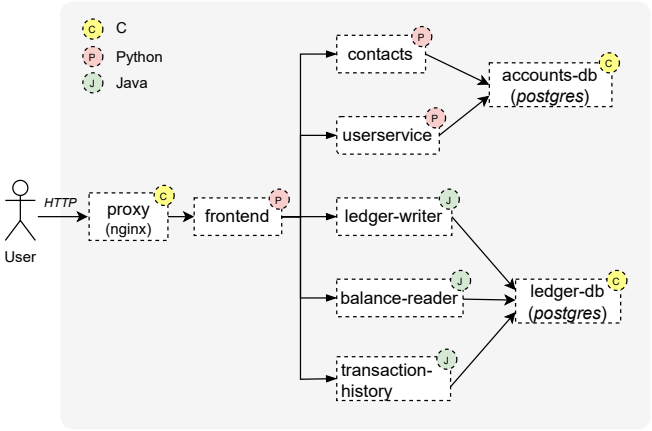



**Figure 4: Bank-of-sirius Architecture**

## 3.1 Use case

In this case study, we allocated one virtual machine (VM) to host the Java-based services and the *ledger-db* service, and another VM for the remaining services. Each service was containerized using Docker to ensure isolated execution environments. We then developed and executed a benchmark script simulating the activities of 100 concurrent users. Each user interacts with the *frontend* service through a specific sequence of HTTP requests: a "/login" request, followed by a "/home" request, and lastly a "/logout" request. It is important to note that the "/home" request is responsible for loading the user's home page, which displays profile information, account balance, and transaction history. Accomplishing this requires the *frontend* service to make sub-requests to the *contacts*, *balance-reader*, and *transaction-history* services. We traced the benchmark execution and used our framework to establish the sequences of thread states involved in the processing of the "/login", "/home", and "/logout" requests. This experiment was repeated tenfold at various time intervals to create a baseline dataset representing application normal performance.

To create our test datasets, we conducted two separate interactions with our target application. For the first, we emulated typical user behavior: logging in, visiting the homepage, and logging out. In the second interaction, we altered the configuration of the *contacts* service by changing the type of Gunicorn workers from *synchronous* to *gThread*. This is a notable change given that all Python-based services in Bank-of-Sirius are Flask applications serviced by Gunicorn servers. Following this modification, we duplicated the initial user behavior. Both interactions were traced, allowing us to capture and compile requests' state sequences into two distinct test datasets.

To identify potential anomalies within the test datasets, our framework scrutinized the captured state sequences for consistency with the established patterns from the baseline dataset. The analysis revealed no anomalies in the first dataset; however, it flagged irregularities in the "contacts" service state sequences in the second dataset. Specifically, we observed that the latencies of this service's spans were unexpectedly lower, and there was a recurrent absence of the "N" (Network) states across numerous spans. This aligns with the variation in Gunicorn operations observed when employing the Sync and gThread models, as illustrated in Fig. 5. Gunicorn uses a pre-fork worker model where a master process manages a set of worker processes dedicated to handling client requests. In the Sync model, each worker handles connections and executes requests one at a time, leading to a simple but less concurrent workflow. Conversely, in the gThread model, each worker pools connections, spawns multiple threads, and distributes tasks across them to efficiently handle simultaneous requests. That explains why the latencies of requests in the latter configuration were lower and the 'N' states were missing from the sequence of states related to the contact service.

## 3.2 Discussion

In this section, we delve into the intricacies of state pattern recognition and its implications for our anomaly detection mechanism. A notable observation was the presence of repetitive pairs of states within the identified patterns, suggestive of specific activities like prolonged network communication or disk read operations (e.g., R-T-R-D-R-D-R-D-R). These repetitive sequences, while indicative of certain behaviors, posed a limitation in the versatility of our pattern-matching algorithm. To enhance our framework's adaptability and precision, we have introduced a transformative step that condenses these repetitive pairs into regular expressions when their occurrence is frequent (e.g., R-T-(R-D)+-R). This refinement not only streamlines the pattern detection process but also enriches our framework's ability to discern more complex behaviors while simplifying the representation of state sequences.

Additionally, we encountered scenarios where the latencies associated with certain states displayed substantial variability, challenging our framework's ability to discern normative from anomalous behavior. For instance, the 'Running' state in a computationally intensive operation, such as factorial computation, can exhibit significant latency fluctuations based on the input magnitude. To address this, we propose the analysis of the latency distributions, particularly for operations marked by a high coefficient of variation. This strategy involves the use of call stack profilers to add a finer granularity to our analysis by accounting for function calls as subspans. This approach allows us to reclassify requests from certain types based on the unique footprint of the executed function calls and their parameters. It would also enable a more tailored detection process that accounts for the distinctive nature of each operation within our microservice architecture.

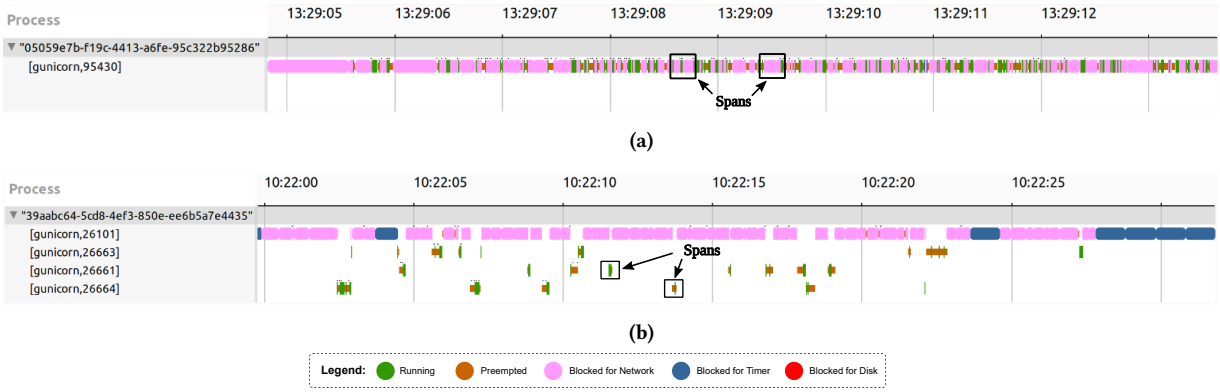



**Figure 5: Critical paths of a single Gunicorn worker in two configurations: Sync (a) and gThread (b). In the Sync configuration, the worker manages both communication and task processing, reflected in running and network states. In contrast, gThread configuration involves the worker handling communication while task processing is distributed among its threads**

## 3.3 Overhead Analysis

Minimizing tracing overhead is essential to prevent skewed results. Excessive overhead may alter the system's normal operation, making the tracing solution impractical for use in production environments. Therefore, to evaluate the overhead incurred by tracing, we conducted performance benchmarks on the Bank-of-Sirius application both with tracing enabled and disabled. We subjected this application to varying workloads and observed its response times. For these tests, we employed Locust [13], an open-source load testing utility, to simulate clients issuing requests at different rates. These clients issue different types of requests to provide a comprehensive view of the system's performance under load. The benchmarking was carried out on a machine equipped with 16 GB of RAM and an Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz. The software environment consisted of Ubuntu 22.04, featuring the 5.15.0-60 kernel version, LTTng 2.12 for tracing, and Docker 24.0.5 for containerization.

The outcomes of our investigation are presented in Fig 6. The latter illustrates the application's response time to "/home" requests—identified as the most resource-intensive requests. Within our test environment, the application under test achieves a peak throughput of 45 requests per second. The graph indicates that the activation of the required tracepoints introduces a negligible performance impact, with tracing causing only a 2 to 4% increase in response time. Interestingly, our benchmark's results also show a slight improvement in the application's response time with tracing enabled at a rate of 5 requests per second. Under identical conditions, this improvement would be unexpected. Nevertheless, a degree of fluctuation is inherent in operating system operations due to many factors such as the scheduling of system processes, and memory page faults. Given that the overhead from tracing a relatively small number of events is almost imperceptible, it becomes challenging to measure and can be smaller than the natural variability of the operating system's performance. In short, our overhead analysis shows that tracing Bank-of-Sirius introduces a marginal increase in its response times, thus confirming the suitability of our framework for production environments.

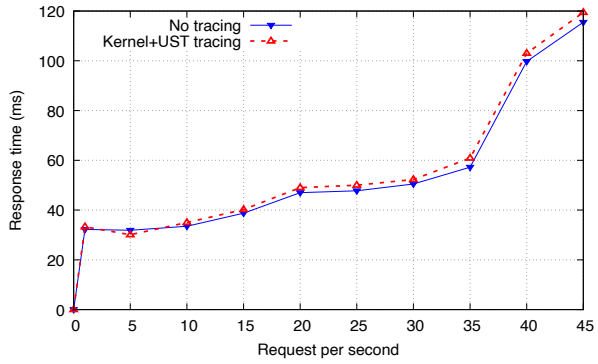



**Figure 6: Bank-of-Sirius's response time to "/home" requests with tracing disabled and enabled**

## 4 RELATED WORK

There is extensive prior work on monitoring and debugging performance problems in distributed and microservice-based applications. Most of it is based on the use of distributed tracing for collecting monitoring data from a distributed system. Distributed tracing indeed provides a broad overview of end-to-end request processing in microservice-based applications. Nonetheless, the information it produces is insufficient to pinpoint the causes of detected latency issues. Many strategies were hence proposed to enrich the span-based traces with data collected from application- and kernel-level logs and tracepoints [3, 26]. For example, authors in [3, 26] proposed an automated instrumentation framework that runs alongside the distributed tracing infrastructure. Their framework combines distributed tracing and variance-based control logic to explore at runtime where logs/tracepoints need to be enabled to effectively help diagnose performance problems. The main limitation of the proposed framework is its incapacity to provide value in diagnosing transient and short-lived performance problems.

On the other hand, various cross-layer tracing techniques have been used in literature to enhance the understanding of distributed

applications behaviors [1, 2, 4, 19]. For example, authors. in [2] inject application-level events into kernel traces by executing a series of innocuous system calls for each high-level event of interest (e.g., the start and end of an RPC call). These system calls serve as synchronization points in the trace to merge high-level and low-level events. Also, in [4], Belkhiri et al. relied on vertical context propagation to inject high-level request identifiers into the kernel. The weakness of this approach is that it poses scalability challenges as it requires a system call each time the target application starts or completes the processing of a request. There are also numerous attempts to diagnose performance anomalies by applying statistics, graph theory, and trace comparison techniques on collected traces [8, 14, 22]. For instance, Huang et al. in [14] leverage the structure within the distributed traces to group similar traces and provide detailed statistics at each level of the trace hierarchy. Their tool can assist practitioners in identifying the relevant operations to focus on when debugging but cannot identify the cause of the issue automatically.

## 5 CONCLUSION

Microservices often complicate the debugging of unexpected latencies in application operations and pinpointing their root causes. This paper addresses this issue by proposing an innovative approach for diagnosing performance anomalies in microservice applications. Our approach leverages cross-layer tracing to enhance the granularity of observability, providing a multi-dimensional view that correlates system resource metrics with user requests. The use of sequential pattern mining enables the isolation of anomalous behavior patterns and facilitates the identification of their root causes. Our evaluations have not only confirmed the efficacy of our framework in identifying performance anomalies but also demonstrated its operational efficiency by maintaining minimal overhead.

As distributed systems evolve, diagnosing performance grows increasingly complex. Our contribution represents a step forward in mitigating this challenge by equipping developers and system operators with a tool capable of identifying and diagnosing performance issues without invasive instrumentation or prohibitive performance penalties. We expect that our findings will incite further research into optimizing distributed tracing infrastructures and developing even more sophisticated analysis techniques. Future work could explore the potential for real-time anomaly detection and automated remediation, which would enhance further the resilience and reliability of microservice-based applications.

## REFERENCES


[1] Marcelo Amaral, Tatsuhiro Chiba, Scott Trent, Takeshi Yoshimura, and Sunyanan Choochotkaew. 2022. MicroLens: A Performance Analysis Framework for Microservices Using Hidden Metrics With BPF. *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)* 00 (2022), 230–240.

[2] Dan Ardelean, Amer Diwan, and Chandra Erdman. 2018. Performance Analysis of Cloud Applications. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, Vol. 00. USENIX Association, Renton, WA, 405–417.

[3] Emre Ates, Lily Sturmann, Mert Toslali, Orran Krieger, Richard Megginson, Ayse K. Coskun, and Raja R. Sambasivan. 2019. An automated, cross-layer instrumentation framework for diagnosing performance problems in distributed applications. *Proceedings of the ACM Symposium on Cloud Computing* (2019), 165–170.

[4] Adel Belkhiri, Ahmad Shahnejat Bushehri, Felipe Gohring de Magalhaes, and Gabriela Nicolescu. 2023. Transparent Trace Annotation for Performance Debugging in Microservice-oriented Systems (Work In Progress Paper). *International Conference on Performance Engineering (ACM/SPEC)* (2023), 25–32.

[5] Yang Cai, Biao Han, Jinshu Su, and Xiaoyan Wang. 2021. TraceModel: An Automatic Anomaly Detection and Root Cause Localization Framework for Microservice Systems. *2021 17th International Conference on Mobility, Sensing and Networking (MSN)* 00 (2021), 512–519.

[6] CNCF. 2023. OpenTelemetry: high-quality, ubiquitous, and portable telemetry to enable effective observability. https://opentelemetry.io/

[7] Mathieu Desnoyers and Michel R Dagenais. 2006. The lttng tracer: A low impact performance and behavior monitor for gnu/linux. In *OLS (Ottawa Linux Symposium)*, Vol. 2006. Citeseer, 209–224.

[8] Prem Devanbu, Myra Cohen, Tao Xie, and Liangfei Su. 2020. Graph-based trace analysis for microservice architecture understanding and problem diagnosis. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2020).

[9] Ericsson. 2023. Trace Compass. http://tracecompass.org/

[10] Dan Feng, Steffen Becker, Nikolas Herbst, Philipp Leitner, Zheng Papadopoulos, Hyunseok Chang, Sarit Mukherjee, and Eric Eide. 2022. LongTale: Toward Automatic Performance Anomaly Explanation in Microservices. *International Conference on Performance Engineering (ACM/SPEC)* (2022), 5–16.

[11] Rodrigo Fonseca, George Porter, Randy H. Katz, and Scott Shenker. 2007. X-Trace: A Pervasive Network Tracing Framework. In *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*. USENIX Association, Cambridge, MA.

[12] Francis Giraldeau and Michel Dagenais. 2015. Wait Analysis of Distributed Systems Using Kernel Tracing. *IEEE Transactions on Parallel and Distributed Systems* 27, 8 (2015), 2450–2461.

[13] Jonatan Heyman, Joakim Hamrén, Carl Byström, and Hugo Heyman. 2023. Locust: An open-source load testing tool. https://locust.io/

[14] Lexiang Huang and Timothy Zhu. 2021. tprof: Performance profiling via structural aggregation and automated analysis of distributed systems traces. *Proceedings of the ACM Symposium on Cloud Computing* (2021), 76–91.

[15] Jaegertracing.io. 2023. Jaeger: Open Source, End-to-End Distributed Tracing. http://jaegertracing.io

[16] Madeline Janecek, Naser Ezzati-Jivan, and Seyed Vahid Azhari. 2021. Container Workload Characterization Through Host System Tracing. *2021 IEEE International Conference on Cloud Engineering (IC2E)* 00 (2021), 9–19.

[17] Jonathan Kaldor, Jonathan Mace, and Yee Jiun Song. 2017. Canopy: An End-to-End Performance Tracing And Analysis System. *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), 34–50.

[18] Iman Kohyarnejadfard, Daniel Aloise, Seyed Vahid Azhari, and Michel R. Dagenais. 2022. Anomaly detection in microservice environments using distributed tracing data analysis and NLP. *Journal of Cloud Computing* 11, 1 (2022), 25.

[19] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-source Data. *arXiv* (2023). arXiv:2302.05092

[20] Chun Li and Jianyong Wang. 2008. Efficiently Mining Closed Subsequences with Gap Constraints. *Proceedings of the 2008 SIAM International Conference on Data Mining* (2008), 313–322.

[21] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. 2020. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)* 00 (2020), 48–58.

[22] Lun Meng, Feng Ji, Yao Sun, and Tao Wang. 2021. Detecting anomalies in microservices with execution trace comparison. *Future Generation Computer Systems* 116 (2021), 291–301.

[23] Inc. NGINX. 2023. Bank of Sirius. https://github.com/nginxinc/bank-of-sirius

[24] Tim Sherwood, Emery Berger, Christos Kozyrakis, Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. Sage: practical and scalable ML-driven performance debugging in microservices. *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2021), 135–151.

[25] Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. *Dapper, a Large-Scale Distributed Systems Tracing Infrastructure.* Technical Report. Google, Inc.

[26] Mert Toslali, Emre Ates, Alex Ellis, Zhaoqi Zhang, Darby Huye, Lan Liu, Samantha Puterman, Ayse K. Coskun, and Raja R. Sambasivan. 2021. Automating instrumentation choices for performance problems in distributed applications with VAIF. *Proceedings of the ACM Symposium on Cloud Computing* (2021), 61–75.

[27] Dean M. Tullsen and Brad Calder. 1998. *Computing along the critical path.* Technical Report. Technical report, University of California, San Diego.

[28] Zhizhou Zhang, Murali Krishna Ramanathan, Prithvi Raj, Abhishek Parwal, Timothy Sherwood, and Milind Chabbi. 2022. CRISP: Critical Path Analysis of Large-Scale Microservice Architectures. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 655—672.

[29] Zipkin.io. 2022. Zipkin. https://zipkin.io


# DMBench: Load Testing and Benchmarking Tool for Data Migration

Fares Hamouda
York University
North York, Canada
faresham@yorku.ca

Marios Fokaefs
York University
North York, Canada
fokaefs@yorku.ca

Dariusz Jania
IBM
Kraków, Poland
dariusz.jania@pl.ibm.com

## ABSTRACT

Data migration refers to the set of tasks around transferring data over a network between two systems, either homogeneous or heterogeneous, and the potential reformatting of this data. Combined with large volumes of data, resource constraints and variety in data models and formats, data migration can be critical for enterprises, as it can consume a significant amount of time, incur high costs, and pose a significant risk if not executed correctly. The ability to accurately and effectively predict these challenges and plan for proper resource, time and budget allocation is vital for the proper execution of data migration. In this work, we introduce the concept of load testing and benchmarking for data migration to allow decision-makers for higher efficiency and effectiveness when planning for such tasks. Our framework aims for extensibility and customizability to enable the execution of a greater variety of tests. Here, we present a prototype architecture, a roadmap of how the development of such a platform should proceed and a simple case study of how it can be used in practice.

## CCS CONCEPTS

• **General and reference** → **Experimentation**; • **Computer systems organization** → **Cloud computing**; • **Information systems** → **Cloud based storage**; **Database performance evaluation**.

## KEYWORDS

data migration; big data; benchmarking; load testing; software performance; data integrity; data transfer

**ACM Reference Format:**
Fares Hamouda, Marios Fokaefs, and Dariusz Jania. 2024. DMBench: Load Testing and Benchmarking Tool for Data Migration. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '24 ), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3653663

## 1 INTRODUCTION

Technological advancements often drive large-scale migrations of software and data systems to new platforms, presenting challenges in data transfer. The volume of enterprise data, ranging from terabytes to petabytes, strains networks and increases the risk of errors. Additionally, diverse target systems may necessitate changes in migration strategies. In a business context, data migration is viewed as a maintenance task that requires resources and careful planning to minimize disruption to regular business operations.

To tackle the planning and resource challenges, our prototype architecture presents a flexible testing platform. It enables swift and adaptable experimentation to systematically assess various migration scenarios and configurations, yielding comprehensive data for informed decisions. Adding to its value, this framework extends its utility to the execution of migration experiments by simulating multiple production workloads on the data source machine. This approach enables a comprehensive assessment of how the workload on the data source machine influences the performance and reliability of the data migration process. Additionally, the framework allows for an investigation into the reciprocal impact—how the migration process affects production workloads and performance on the data source. We maintain that this prototype's inherent flexibility can readily accommodate such complex scenarios, positioning it not only as a benchmarking tool but also as a versatile resource for in-depth migration experimentation and analysis.

In this work, we outline the main components of the prototype architecture while highlighting essential non-functional requirements crucial for platform design. We assert that the prototype's inherent flexibility allows for the accommodation of more complex migration scenarios and configurations. Furthermore, we envision this platform evolving beyond its benchmarking capabilities to serve as a dynamic tool for data collection, decision-making, or even as the foundation for a dynamically adaptive migration strategy.

## 2 RELATED WORK

Data migration has mainly been studied in the context of migration to the cloud [6, 9, 15]. Some studies investigate methods for validating data migration to ensure data accuracy post-migration. These approaches include comparing data sets migrated using different protocols, validating migration based on comparison data, and employing techniques such as checksums or key-data pairs [7, 10, 11, 16]. Additionally, research on optimizing the migration process focuses on minimizing testing costs, customizing migration problems with specific constraints, and proposing streamlined algorithms [2, 4, 5, 8]. Besides practitioners, our platform can also support this research by allowing testing of novel validation and optimization techniques. Subramani et al. [13] propose a theoretical algorithm to optimize data migration in order to minimize

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE Companion '24 , May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3653663

testing on the application side. However, they do not discuss any method or tool for testing the actual migration.

# 3 DMBENCH ARCHITECTURE AND IMPLEMENTATION

DMBench is designed to be primarily functional, usable and portable, but also flexible and extensible. To this end, it consists of multiple semi-independent modules with distinct roles that can be easily configured individually or as a whole, and can be replaced by alternative implementations with ease. The use of Docker containers makes the platform easy to deploy and redeploy at will and on demand. Next, we provide more details about the properties and relations between these modules.

## 3.1 Functional and Non-Functional Requirements

The primary objective of DMBench is to allow testers and decision-makers to design and execute migration experiments as efficiently and as effectively as possible. Such a platform should allow for easy configuration of different experiments, fast deployment and execution of the experiment and fully automated and comprehensive presentation of the results. In principle, the platform should enable testers to execute as many "what-if" scenarios as possible to compare many alternatives or confirm many hypotheses. In practice, the platform guides the tester to prepare an environment to send data from one machine to another (potentially remote) with different types of configurations, and then monitors each part of the process.

DMBench has been designed according to a set of principles on the functional and non-functional requirements of a data migration testing tool, as described below.

### Functional Requirements:

- **Easy setup:** The setup for a migration task refers to all steps relevant to preparing the source and target machines, the migration engine, any metering apparatus (e.g., logging or monitoring) along with respective databases to save results, and the controller of the migration test or experiment. Setup can be a time-consuming step, especially when it is not automated. The ability to quickly complete the setup and make it easy to share between experiments is vital to the ability of testing multiple scenarios and configurations.
- **Easy configuration:** Numerous parts of the platform and of an experiment need to be configurable to allow for extensive control to the tester. If the configuration is complex, inconsistent or generally lacking, it may render experimentation results unusable.
- **Convenient and complete access to results:** For any benchmark to be useful, it needs to provide the results in a convenient way to enable further analysis and interpretation. While this can include reports with visualizations and descriptive statistics, it is important to also return all raw measurements and results in a format that can be easily digested by an analysis software.

*Non-functional Requirements.*

- **Usability:** The user of the tool is expected to have some basic understanding of data migration and potentially of the source and target systems, and the data to be transferred. Besides that, the tools should hide as many details about the experimentation infrastructure as possible and expose any configuration or input points through a clear and easy-to-use interface. The proposed benchmark follows a simplified approach, allowing users to initiate experiments with a few straightforward commands after the environment is set up and configured. The objective is to streamline the process, ensuring ease of use while maintaining simplicity.
- **Extensibility:** The benchmark as a software framework. Through the configuration files, the user can provide outside input to the framework and provide any migration engine she wants to test by dockerizing it following our process. In addition, all other modules, including logging and monitoring, can be replaced by what the user desires, as long as they can be deployed on Docker. Besides, the "frozen spots" of the framework, i.e., the abstract components or the components that the user cannot override, dictate the flow of the experiment.
- **Accuracy and Reliability:** When configuring an experiment, the user has the possibility to request for each experiment to be executed a given number of times. By repeating the exact same experiment multiple times and averaging over the results of the iterations, we can ensure that any source of variability is excluded, and the results are accurate and reliable. In many analyses, it is required to perform statistical testing to confirm or reject our initial hypothesis. By repeating the experiments multiple times, we make it possible to run such tests with high confidence. No matter the number of repetitions, experiments in DMBench are executed deterministically and any variations originates from the environment or the use case and not from the tool. Through complete access to the results, this is verifiable by the tester.

## 3.2 Architecture

Figure 1 shows the main components of DMBench, which are described in detail next.

- **Migration Engine:** Central to the framework, the Migration Engine serves as the focal point for all bench-marking activities, but is primarily the module responsible for transferring data from the source to the target. All other components within the framework are designed to closely monitor and assess the performance of the engine. Users are afforded the flexibility to select any migration engine of their preference for bench-marking purposes. Whether the chosen engine operates within a singular service architecture or spans multiple services, the only prerequisite is the dockerization of the selected engine. The framework seamlessly manages the intricacies of the bench-marking process, offering a streamlined and user-friendly experience.
- **Data Source:** This component serves as the starting point for the Migration Engine to transfer data from here to a designated target machine. DMBench requires only an IP address and appropriate credentials to connect to the Data

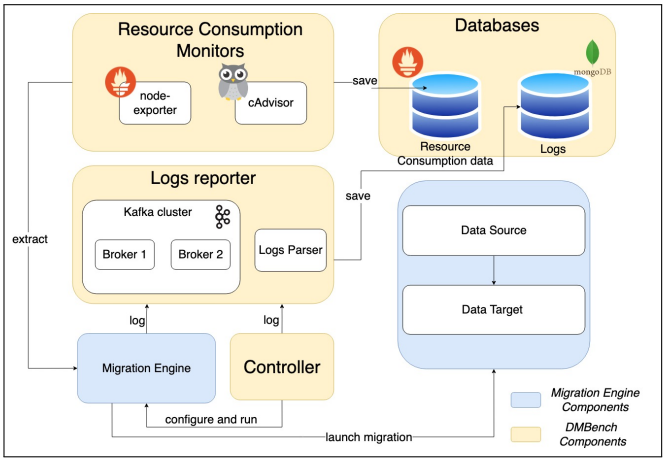



**Figure 1: The general architecture of DMBench.**

Source machine, but the specific deployment method (virtual machine, physical server, container) is irrelevant to the platform.

- **Data Target:** This component is the destination for data transferred by the Migration Engine from the source machine. Similar to the sources, only an IP address and connection credentials are necessary.
- **Controller:** Responsible for orchestrating all experiments under consistent conditions, the Controller configures the migration engine for each experiment with varying parameters. It initiates and oversees the execution of the migration process, monitoring the engine's performance through recording and analysis of migration logs. Simultaneously, the Controller tracks resource consumption by deploying `cAdvisor` [14]and `node-exporter` [12] on the migration infrastructure.
- **Metrics & Logs Databases:** DMBench uses two databases for the monitored data. The first, a time series database based on `Prometheus` [1], aggregates resource consumption data collected from `cAdvisor` [14]. The second database, a `MongoDB` [3] instance, serves as the repository for all log data generated during the experiments. These databases work in tandem, with `Prometheus` [1] focusing on resource metrics and the second database storing all logs from both the framework and the migration engine, it ensures comprehensive and organized storage of experiment results.
- **Logs reporter:** Comprising two integral components, the Logs Reporter ensures a robust and organized handling of experiment logs. The first component involves a `Kafka` cluster, serving as the repository for all logs. Both the Controller and the Migration Engine publish their logs to `Kafka`, with a dedicated consumer responsible for retrieving and temporarily storing these logs in local files.
  The second component is the parser, which not only extracts data from the logs but also transforms it into a human-readable format. The parsed information is then exported into CSV files before being permanently stored in a `NoSQL` database, as mentioned above. This dual-component approach ensures a seamless and efficient process for managing, interpreting, and extracting insights from the experiment logs.

In our framework, the configuration is specified in a config.ini format, which is parsed within the framework. The framework ensures that the config.ini file is available in a specified path within the Docker container where the migration engine is deployed. The engine then reads the config.ini file to run the experiment based on the provided values. While this approach has been tested with our engine and the DB2 migration engine using a simple Python script to read and execute the configuration, it's important to highlight that the specifics of handling the configuration may vary depending on the engine being used. Therefore, users are responsible for implementing the necessary scripts or procedures to ensure compatibility with their chosen engine. For detailed config.ini explanations, including section meanings and purposes, visit the GitHub repository[1], these details are found in the controller configuration section.

## 4 CURRENT IMPLEMENTATION STATE

DMBench can presently accommodate: a) a proprietary **default migration engine**, designed for migrating files from a source to a target machine, b) a **MySQL database migration engine**, where the database is first dumped into a file and subsequently migrated using our default migration engine, and c) the **IBM DB2 migration engine**, which is discussed below. To facilitate the adoption of DMBench, comprehensive guidelines on its utilization and further development are meticulously documented and available in a GitHub repository[2]. These guidelines provide users with step-by-step instructions and best practices, ensuring a smooth and efficient migration process.

In addition to the various migration engines, the framework supports a number of different migration scenarios in terms of size and format of data: a) Exploring multiple compression techniques (e.g., GZIP, LZ4) or migrating data over multiple streams, to optimize migration engine configurations. b)The framework enables the assessment of migration engine limitations, such as maximum stream capacity or data volume, by incrementally increasing parameters until system constraints are reached. c) A key feature of the framework is its capability to compare multiple migration engines using predefined datasets, allowing for a comparative analysis of their performance characteristics during the migration process.

### 4.1 Case Study: IBM DB2 Migration

One practical use case in our framework focuses on migrating data between two IBM Db2 databases. This migration is facilitated by IBM's dedicated migration service, initially containerized using Docker. Following the guidelines detailed in the documentation of our GitHub repository, we establish the required environment. Furthermore, the Controller is configured based on the specifications provided in the documentation.

(1) Ensuring the requisite machines are set up and accessible, we've verified the readiness of both the source and target machines, which are IBM Db2 databases.

[1]https://github.com/yorku-ease/DataMigrationBenchmarkingTool?tab=readme-ov-file#configuration
[2]https://github.com/yorku-ease/DataMigrationBenchmarkingTool

(2) In the configuration phase, we focused on configuring various components of the framework. Notably, the configuration of the Controller is detailed in the Table 1.
(3) Subsequently, the experiment was executed following the steps outlined in the documentation, and the results were obtained and stored in both MongoDB [3] and Prometheus [1].

| Key | Value |
|---|---|
| **Section : targetServer** | |
| host | 192.168.122.52 |
| username | db2inst1 |
| password | password |
| port | 50000 |
| type | db2 |
| **Section : sourceServer** | |
| host | 192.168.122.28 |
| username | db2inst1 |
| password | password |
| port | 50000 |
| **Section : KafkaCluster** | |
| host | 192.168.122.145 |
| port | 9092 |
| performanceBenchmarkTopic | performanceBenchmark |
| migrationEngineTopicName | migrationEngine |
| frameworktopicname | framework |
| **Section : migrationEnvironment** | |
| migrationEngineDockerImage | fareshamouda/d-b2migrationservice |
| loggingId | |
| numberofexperiments | 1 |
| **Section : experiment** | |
| compress | NO,GZIP,LZ4 |
| maxStreams | 3 |
| sourceDatabasetoTargetDatabase | sample=>testdb |
| tables | DEPARTMENT |

**Table 1: Configuration parameters passed to the controller for the IBM Db2 case study.**

## 5 FUTURE DEVELOPMENT

The primary aim of the benchmark is to generate data for decision-making and to enhance understanding of migration tasks and systems. In this context, DMBench will be used to accumulate a significant data and knowledge base on the performance of data migration tools and tasks under a large variety of migration scenarios. This database will then be used to develop performance models, enabling simulations and faster decision-making. In addition, DMBench will be extended to support more migration engines to allow for comparisons and effective choice making for practitioners.

## 6 CONCLUSION

DMBench is a flexible framework for testing the performance of data migration engines. Through systematic setup, configuration, and execution, the framework proves its adaptability to diverse migration scenarios. Leveraging Docker technology and robust logging, it efficiently captures performance benchmarks and resource consumption data.

# REFERENCES


[1] Julius Volz and Björn Rabenstein and Matt Bostock. 2012. *Prometheus : an open-source monitoring and alerting toolkit.* SoundCloud. https://prometheus.io/

[2] Eric Anderson, Joe Hall, Jason Hartline, Michael Hobbs, Anna R. Karlin, Jared Saia, Ram Swaminathan, and John Wilkes. 2001. An Experimental Study of Data Migration Algorithms. In *Algorithm Engineering*, Gerth Stølting Brodal, Daniele Frigioni, and Alberto Marchetti-Spaccamela (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 145–158.

[3] Dwight Merriman, Eliot Horowitz, and Kevin Ryan. 2007. *MongoDB: an open-source, document-oriented NoSQL database.* DoubleClick. https://www.mongodb.com/

[4] M Elamparithi and V Anuratha. 2015. A Review on Database Migration Strategies, Techniques and Tools. *World Journal of Computer Application and Technology* 3, 3 (2015), 41–48.

[5] Zhao JF. and Zhou JT. 2014. Strategies and Methods for Cloud Migration. *International Journal of Automation and Computing* 11 (2014), 143–152.

[6] Kevin Kline, Denis McDowell, Dustin Dorsey, and Matt Gordon. 2022. Moving Your Data to the Cloud. In *Pro Database Migration to Azure: Data Modernization for the Enterprise.* Springer, Berlin, Germany, 263–283.

[7] TN Manjunath, Ravindra S Hegadi, and HS Mohan. 2011. Automated data validation for data migration security. *International Journal of Computer Applications* 30, 6 (2011), 41–46.

[8] Johny Morris. 2012. *Practical data migration.* BCS, The Chartered Institute, London, United Kingdom.

[9] Stephen Orban. 6. Strategies for Migrating Applications to the Cloud. *Medium. Library Catalog: medium. com* 6 (6).

[10] PR Devale P Paygude. 2013. Automated Data Validation Testing Tool for Data Migration Quality Assurance. *International Journal of Modern Engineering Research (IJMER)* 3 (2013), 599–603.

[11] Priyanka Paygude and PR Devale. 2013. Automation of data validation testing for QA in the project of DB migration. *International Journal of Computer Science* 3, 2 (2013), 15–22.

[12] Prometheus community. [n. d.]. *Node Exporter: a software component used in conjunction with Prometheus for monitoring Linux and UNIX system.* https://github.com/prometheus/node_exporter

[13] K. Subramani, Bugra Caskurlu, and Alvaro Velasquez. 2019. Minimization of Testing Costs in Capacity-Constrained Database Migration. In *Algorithmic Aspects of Cloud Computing*, Yann Disser and Vassilios S. Verykios (Eds.). Springer International Publishing, Cham, 1–12.

[14] Google Core Team. 2014. *cAdvisor: an open-source container monitoring and performance analysis tool.* Google. https://github.com/google/cadvisor

[15] Jinesh Varia. 2010. Migrating your existing applications to the aws cloud. *A Phase-driven Approach to Cloud Migration* (2010), 1–23.

[16] Bin Wei and Tennyson X Chen. 2014. *Verifying Data Migration Correctness: The Checksum Principle.* RTI Press, United States.


# STIGS: Spatio-Temporal Interference Graph Simulator for Self-Configurable Multi-Tenant Cloud Systems


Iqra Zafar*
Hasso Plattner Institute
University of Potsdam
iqra.zafar@hpi.de

Christian Medeiros Adriano
Hasso Plattner Institute
University of Potsdam
christian.adriano@hpi.de

Holger Giese†
Hasso Plattner Institute
University of Potsdam
holger.giese@hpi.de



## ABSTRACT

The finer-granularity of microservices facilitate their evolution and deployment on shared resources. However, resource concurrency creates elusive interdependencies, which can cause complex interference patterns to propagate in the form of performance anomalies across distinct applications. Meanwhile, the existing methods for Anomaly Detection (*AD*) and Root-Cause Analysis (*RCA*) are confounded by this phenomenon of interference because they operate within single call-graphs. To bridge this gap, we develop a graph formalism (Spatio-Temporal Interference Graph - *STIG*) to express interference patterns and an artifact to simulate their dynamics. Our simulator contributes to the study and mitigation of interference patterns as a performance phenomenon that emerges from regular resource consumption anomalies.



## CCS CONCEPTS

• **Computer systems organization → Distributed architectures**;

## KEYWORDS

Microservices, Anomaly Propagation, Interference, Multi-tenant Cloud Systems, Self-Configuration

**ACM Reference Format:**
Iqra Zafar, Christian Medeiros Adriano, and Holger Giese. 2024. STIGS: Spatio-Temporal Interference Graph Simulator for Self-Configurable Multi-Tenant Cloud Systems. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '24 ), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3653664


## 1 INTRODUCTION

In the ever-evolving landscape of cloud computing, microservices have emerged as a dominant architectural style, enabling more flexible and scalable applications. This style relies on a finer-granularity of functions and more radical resource sharing among different applications. However, this strategy increases overall system complexity by adding elusive interdependencies among microservices [6] from distinct applications.

*Definition 1.1.* **Interference** happens when two services that have no logical dependency (caller-callee relation) compete for the same resource (compute, memory, I/O) to the extent that they affect each other's performance (e.g., throughput, latency) [9].

Contrary to the caller-callee relations [5], in application call-graphs and abstract syntax trees, these new interference-enabling interdependencies are more elusive because their presence and flow of direction are not deterministic. Instead, interdependencies might appear and disappear according to the non-stationary patterns of the applications' usage and the work of load balancing or self-configurable service placement mechanisms. Therefore, cross-application services interference confounds the outcome of traditional microservice diagnostic methods like Anomaly Detection (*AD*) and Root-Cause Analysis (*RCA*) [4, 11], as these methods rely on stable and predictable call-graph dependencies [5].

While self-configuration solutions can dynamically adapt to changes in the application usage [3], multi-tenant systems require more involved approaches [10]. For that, various interference mitigation (*IM*) methods have been developed - originally, for virtualized cloud environments [9] and, lately, for microservices [1, 7]. Nonetheless, there are still at least two obstacles that prevent existing *IM* methods from reducing confounding in *AD* and *RCA* approaches: (1) limited number of covered services (four as in [1, 12]), and (2) reliance on metrics that are agnostic to the interdependencies across applications. These methods measure interference w.r.t. *sensitivity* (the susceptibility of a service to be influenced by other services) and *contention* (the service consumption demand on a resource, e.g., CPU) between service pairs, but they are oblivious of the many-to-many relationship nature of interference.

Conversely, our approach overcomes these limitations by formulating the interference phenomenon as a spatio-temporal graph. Our corresponding simulation helps mitigate the probability and impact of the interference phenomenon by de-confounding the diagnostics from the *AD*, *RCA*, and *IM* methods, hence, rendering these methods more effective for complex multi-tenant cloud systems [1, 12]. We contribute with (1) a **formalism** to capture interference patterns as spatio-temporal graphs (*STIG*), (2) a **simulator** called STIGS (Figure 2) for generating interference patterns, and (3) a practical **evaluation** with three popular microservice benchmarks (Bookinfo[1], TeaStore[2] and SockShop[3]).

*Definition 1.2.* **Spatio-Temporal Interference Graph (*STIG*)** is denoted as $\mathcal{G} = (V, E, X_{v(t)}, X_{e(t)})$, where $V$ are nodes representing services, $E$ are directed edges representing interference between

*ACM Member
†IEEE and ACM Member



This work is licensed under a Creative Commons Attribution International 4.0 License.



*ICPE Companion '24 , May 7–11, 2024, London, United Kingdom*


© 2024 Copyright held by the owner/author(s).


ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3653664


[1]Bookinfo:https://github.com/nocalhost/bookinfo
[2]Tea-Store: https://github.com/DaGeRe/TeaStore
[3]SockShop: https://microservices-demo.github.io/

services across applications, $X_{v(t)}$ are the time-varying node features (e.g., resource per service), and $X_{e(t)}$ the edge features (e.g., interference probability).

## 2 INTERFERENCE ANOMALY SCENARIO

As an example, assume three e-commerce applications having 14 microservices (shown in Figure 1) deployed on the same server (either *host1*, *host2*) each with a CPU of 4 cores and 10 GB of memory. The occurrence of a sudden surge of 100% in users during a flash sales event could subsequently cause an increase in the demand for these applications, e.g., from 60% to 90% CPU and memory usage from 5GB to 10GB. As the services compete for shared resources, the increased load could induce a low response time, e.g., 1000ms from the original 100ms among the resource-sharing services. This, in turn, could evolve to more severe problems like intermittent or permanent failures. Because anomalies jump across the applications' borders, one cannot rely on the individual call-graphs and performance metrics. To address this situation, the *STIG* model captures the dependencies originating both from the call-graph and the deployment graph (e.g., service placement configuration).

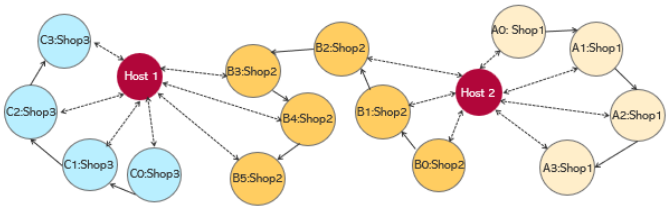



**Figure 1: Knowledge Deployment Graph. Nodes colors for distinct applications (shops) and maroon/red color for host nodes. The dashed arrows for hosting service relationships and the solid arrows for caller-callee relationships.**

## 3 STIG SIMULATOR

### 3.1 Design and Architecture

The workflow of the STIGS depicted in Figure 2 represents a structured approach to modeling and analyzing interference in multi-node applications, which we detail next. The task **Define Multi-Node Application** generates the dependency graphs from the system architecture (*System Archi.xml*) and the deployment configuration (*Deployment config.yaml*). Based on that, we **Instantiate the Semantic Model Template** to extract distinct interference-enabling paths. The **Graph Generator** combines the set of distinct paths and the multi-tenant setup (*Deployment config.yaml*) to generate (1) a knowledge deployment graph (e.g. Figure 1) and (2) the time-annotated call-graphs, which serve as ground truth for the *STIG* generation process. The **Impacted Pair Generator** task identifies the candidate pairs of service nodes with the potential for mutual interference. The **Interference Probability Calculator** estimates the likelihood of interference by taking into account both the execution timings and their history of service anomalies. Finally, one or multiple instances of the *STIG* (e.g. Figure 3) are generated to represent distinct likelihood scenarios of anomalies induced by interference between services across applications. If at least one *STIG* was generated, the workflow ends and the simulator proceeds to **Graph Display**, where the *STIG* set is made available for analysis. We provided detailed instructions on how to install the STIG Simulator which is available for download on Zenodo[4] and Github[5].

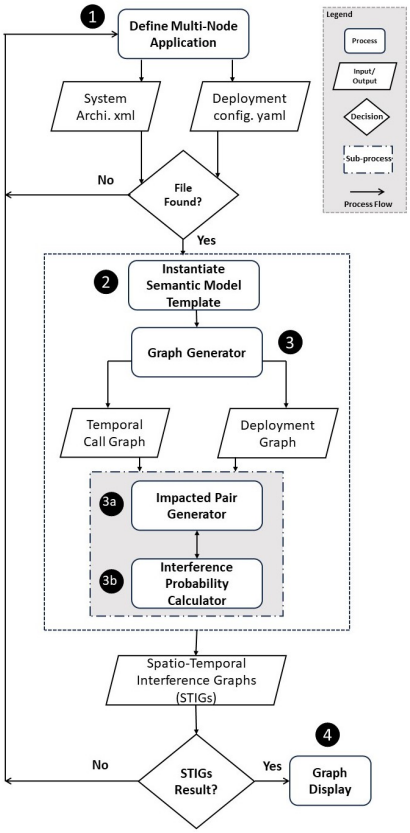



**Figure 2: *STIG* Simulator Workflow**

### 3.2 Algorithms

To investigate the interference phenomenon, we identify the source and corresponding impact of the interference through the proposed algorithms. In Algorithm 1, we computed query predicate stack that acts as sources and targets of interference, respectively, from the Knowledge Deployment Graph (*kgraph*) and particular host node (*Host1* in Figure 1). These stack computations depend on the execution order of calls at the specific host (**line 5 and 10**). The source of interference on one or more targets services is capture as a probability measure proportionate to the magnitude of shared resources within a time window. Consequently, longer time intervals and higher resource utilization entail higher probability of interference (computed by the Algorithm 2). This involves generating a list of the impacted node pairs (*sourceStack* and the *targetStack*) based on their execution overlapping times. The algorithm first sorts these stacks by their execution start time (**line 2**) and matches the current

[4]Zenodo repository: https://zenodo.org/records/10610874
[5]https://github.com/christianadriano/STIGS-Artifact

source node and the target nodes list given their execution time conditions (**lines 3-10**). The probability of interference is derived for each source node (*curSource*) and their respective overlapping target nodes (*curTargetList*), also factoring-in their levels of shared resource usage. With that, we can estimate the interference probabilities for the *STIG* (**line 12** by calling Algorithm 3). This involves computing for each source node (*curSource*) the list of target nodes (*curTargetList*) and their corresponding execution time overlap, as well as the magnitude of the resource usage shared with each source and target nodes (**lines 3-6**). The resulting list of impacted pairs is then returned by Algorithm 2 (**line 15**).

**Algorithm 1** Compute Query Predicate Stack

```
1:  procedure GENERATEQPSTACK(kgraph, host, filepath)
2:      arch = get.architecture.callgraph(kgraph)
3:      deploy = get.deploy.callgraph(kgraph)
4:      if file at filepath exists then
5:          exeOrders = Load data from filepath
6:      else
7:          exeOrders = createTempgraph(arch, filepath)
8:      end if
9:      serPaths = getDistPaths(archi)
10:     nodes.at.host = List all nodes deployed on host
11:     exe.orders= exe orders in nodes at host
12:     Initialize Query.Predicate.stack as an empty list
13:     for each exe.order in exe.orders.at.host do
14:         Get index of first service path in serPaths
15:     end for
16:     for each ser in exe.orders do
17:         Create stack.entry with service details
18:         if ser on same path of service in exe.orders then
19:             Add stack.entry to query
20:         else
21:             Add stack.entry to predicate
22:         end if
23:     end for
24:     Update start time for each entry in query and predicate lists
25:     Add a dictionary with query and predicate to
    query.predicate.stacks
26:     return query.predicate.stacks
27: end procedure
```

Using this information, we can construct Spatio-Temporal Interference Graphs (STIGs) as described in Algorithms 1,2 and 3. The STIG, as seen in Figure 3, consists of nodes as services, solid edges as service calls within the same application, and the dotted edges standing for interference paths. The weights on the interference edges can be initialized with prior probabilities based on temporal execution overlap across application services sharing the same resource (worker-node).

## 4 EVALUATION CASE STUDY

We deploy three popular benchmarks (**BookShop**, **TeaShop**, and **SockShop**) on a Kubernetes cluster and generate traces by injecting requests (10 to 1000) to their front webpages. Traces are collected based on the following Table 1 configurations. The "Number of

**Algorithm 2** Compute List of Impacted Pairs

```
1:  procedure IMPACTEDPAIRLIST(sourceStk, targetStk)
2:      Sort both input sets by start of execution
3:      while sourceStk is not empty do
4:          Pop curSource from sourceStk
5:          while endTime of curSource > starting.time.target at
    head of targetStk do
6:              Pop curTarget from targetStk
7:              Put curTarget into curTargetList
8:              if endingTime of curSource is < ending time of
    curTarget then
9:                  Set starting time of curTarget to endingTime of
    curSource
10:                 Push it back to stack
11:             end if
12:             Append ComputeSTIGProb (curSource,
    curTargetList) to resultList
13:         end while
14:     end while
15:     return resultList
16: end procedure
```

**Algorithm 3** Compute STIG Interference Probability Edges

```
1:  procedure COMPUTESTIGPROB(curSource,curTargetList)
2:      totalSourceT=curSource.endTime-curSource.startTime
3:      for each node in curTargetList do
4:          totalTargetT=min(curTarget.endTime,
    curSource.endTime)-curTarget.startTime
5:          curMag=curSource.resUsage+curTarget.resUsage
6:          return {source, target, prob, mag}=curSource,
    curTarget, totalTargetT / totalSourceT, curMag
7:      end for
8:  end procedure
```

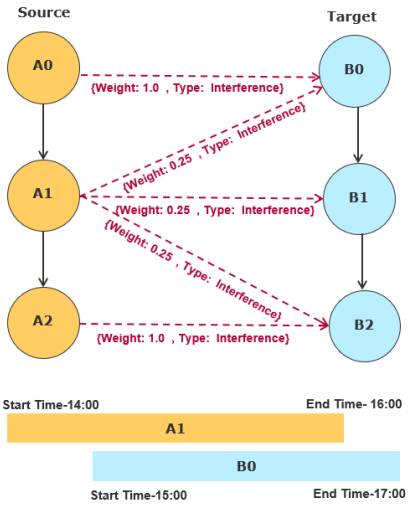



**Figure 3: Spatio-Temporal-Interference-Graph. (*STIG*). Nodes are services, solid edges are calls within one application, and the dotted edges are interference paths**

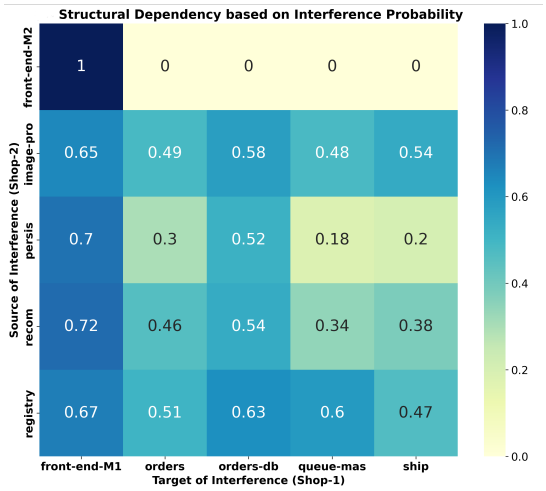



**Figure 4: Structural Dependency Matrix: consolidates the averages of interference across a *STIG* set.**

Requests" column shows how many requests are made in each configuration. This starts at 10 requests in *config1* and increases progressively, reaching up to 1000 requests in *config11*. The "Rate" of request is every 1 min. These generated traces will help in our analysis in combination with STIGs. Traces dataset is available at simulator's Github repository.

**Table 1: Configuration of Traces Generation**

| Config | of Requests | Rate |
|---|---|---|
| config1 | 10 | every 1 min |
| config2 | 20 | every 1 min |
| config3 | 30 | every 1 min |
| ... | ... | ... |
| config10 | 800 | every 1 min |
| config11 | 1000 | every 1 min |

## 4.1 STIG Analysis

To visualize the cause-effect phenomenon on generated STIGs, we extracted only the source and target pairs of the front-end service based on the maximum interference effect and obtained all associated source and target pairs. As a reference, Figure 4 shows a *structural dependency matrix* (*SDM* [2]) representing the interference probabilities (*STIG* edges) between source and target services (*STIG* nodes) of **SockShop** and **TeaShop**, where the darker colors represent higher probability. In the *SDM*, the *front-end-M1:shop1* shows the highest probability (1.0) of being interfered with by *front-end-M2:shop2*, which stems from the assumed determinism of these services starting simultaneously. Conversely, as the effect of interference propagates, there is a lower interference probability, which reflects smaller execution overlap between downstream services.

## 4.2 Reconfiguration Plan

The reconfiguration plan involves ranking the services with respect to the highest probability of necessity and sufficiency of being the culprit of the anomaly induced by interference. Because interference happens both ways, the plan can attribute source and target to anomalous services in either side of an interference association. For this, we monitored and collected traces from shops (Table 1) and performed probabilistic analysis on them. Probability of Necessity (*PN*) consists of the chance that an effect (anomaly) on a target node ($Y = 1$, i.e., $Y$) is caused (interfered) by an anomaly on a source node ($X = 1$, i.e., $X$), given that there is a history of absence of anomaly on the target node ($Y = 0$, i.e., $Y'$) and there is an absence of anomaly on the source node ($X = 0$ or $X'$). Formally, from Pearl [8], *PN(Y,X) = P(Y,X|Y',X')*. The Probability of Sufficiency (*PS*) is the reverse case *PS(Y,X) = P(Y',X'|Y,X)*, while the probability of both Necessity and Sufficiency (*PNS*) is the weighted average *PNS(Y,X) = P(X,Y)PN(Y,X) + P(X',Y')PS(Y,X)*. Among the various approaches to compute these probabilities, we adopted the formulations in [8] (section 19.3.3) that assume causal *exogeneity*[6] and *monotonicity*[7]. The formulations are the following *PNS = P(Y|X) - P(Y|X')*, *PN = PNS / P(Y|X)*, and *PS = PNS / [1 - P(Y|X')]*. The results in Table 2 show that *PN* is more than two orders of magnitude higher than *PS* and *PNS*. This means that one can focus primarily on tackling the *necessary* sources of the induced anomaly, i.e., *product-page* and *reviews*. To mitigate the interference-induced anomalies on *teastore-webui*, one could reconfigure the deployment graph in a way that this microservice is placed on a *worker-node* where there are no instances of the *product-page* and *reviews* microservices. Meanwhile, the *STIG* simulated data also informs us that the other anomalies (e.g., on (*teastore-auth* and *teastore-image* services) are not induced by an interference. For these cases, the solution is to add more resources (compute, memory) to their corresponding worker-nodes. For more details, an analysis is available under the artifact Github repository data/traces folder[8].

**Table 2: Results for two interfering service pairs[a]**

| Item | Pair 1 | Pair 2 |
|---|---|---|
| $X$ | product-page | reviews |
| $Y$ | teastore-webui | teastore-webui |
| *PN* | 8.40% | 24.97% |
| *PS* | 0.05% | 0.05% |
| *PNS* | 0.05% | 0.05% |

[a] The only services with anomalies in **BookShop** are *product-page* and reviews, whereas in **TeaShop** only the *teastore-webui*, *teastore-auth*, and *teastore-image* have anomalies. However, there were only two pairs of services with joint probabilities $P(Y, X) > 0$ (shown in the table).

# 5 CONCLUSION AND FUTURE WORK

We presented a novel approach to the problem of service interference in multi-tenant microservice architectures, where concurrency over shared resources induces the propagation of elusive anomaly patterns. Our formalism and simulator are a contribution to the study of interference anomalies and the mitigation of this complex emergent phenomenon. The artifact components and the interference simulation can be easily extended to new performance anomaly scenarios. In future work, we plan to study the scalability and latency of the simulator within larger and more heterogeneous deployments.

[6] *Exogeneity* = no hidden confounders beyond the detected anomalies
[7] Monotonicity of the causal effects, i.e., anomalies cannot cancel each other.
[8] Analysis Details: https://github.com/christianadriano/STIGS-Artifact/blob/main/data/traces/excel-filtered-combined-services-anomalies.xlsx

# REFERENCES


[1] Madhura Adeppady, Paolo Giaccone, Holger Karl, and Carla Fabiana Chiasserini. 2023. Reducing Microservices Interference and Deployment Time in Resource-constrained Cloud Systems. *IEEE Transactions on Network and Service Management* (2023). https://doi.org/10.1109/TNSM.2023.3235710

[2] Tyson R Browning. 2015. Design structure matrix extensions and innovations: a survey and new opportunities. *IEEE Transactions on engineering management* 63, 1 (2015), 27–52.

[3] Vincent Bushong, Amr S. Abdelfattah, Abdullah A. Maruf, Dipta Das, Austin Lehman, Eric Jaroszewski, Michael Coffey, Tomas Cerny, Karel Frajtak, Pavel Tisnovsky, and Miroslav Bures. 2021. On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study. *Applied Sciences* 11, 17 (2021). https://doi.org/10.3390/app11177856

[4] Shenghui Gu, Guoping Rong, Tian Ren, He Zhang, Haifeng Shen, Yongda Yu, Xian Li, Jian Ouyang, and Chunan Chen. 2023. TrinityRCL: Multi-Granular and Code-Level Root Cause Localization Using Multiple Types of Telemetry Data in Microservice Systems. *IEEE Transactions on Software Engineering* (2023).

[5] Devki Nandan Jha, Saurabh Garg, Prem Prakash Jayaraman, Rajkumar Buyya, Zheng Li, and Rajiv Ranjan. 2018. A holistic evaluation of docker containers for interfering microservices. In *2018 IEEE International Conference on Services Computing (SCC)*. IEEE, 33–40.

[6] Claus Pahl, Pooyan Jamshidi, and Olaf Zimmermann. 2018. Architectural principles for cloud software. *ACM Transactions on Internet Technology (TOIT)* 18, 2 (2018), 1–23.

[7] Yicheng Pan, Meng Ma, Xinrui Jiang, and Ping Wang. 2021. Faster, deeper, easier: crowdsourcing diagnosis of microservice kernel failure from user space. In *30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 646–657.

[8] Judea Pearl. 2022. Probabilities of causation: three counterfactual interpretations and their identification. In *Probabilistic and Causal Inference: The Works of Judea Pearl*. 317–372.

[9] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, and Calton Pu. 2010. Understanding performance interference of I/O workload in virtualized cloud environments. In *2010 IEEE 3rd international conference on cloud computing*. 51–58. https://doi.org/10.1109/CLOUD.2010.65

[10] Miguel G Xavier, Kassiano J Matteussi, Fabian Lorenzo, and Cesar AF De Rose. 2016. Understanding performance interference in multi-tenant cloud databases and web applications. In *2016 IEEE international conference on big data (big data)*. IEEE, 2847–2852.

[11] Ruyue Xin, Peng Chen, and Zhiming Zhao. 2023. Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications. *J. of Systems and Software* (2023).

[12] Chaobing Zeng, Fangming Liu, Shutong Chen, Weixiang Jiang, and Miao Li. 2018. Demystifying the Performance Interference of Co-Located Virtual Network Functions. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 765–773. https://doi.org/10.1109/INFOCOM.2018.8486246

# KubePlaybook: A Repository of Ansible Playbooks for Kubernetes Auto-Remediation with LLMs

Komal Sarda*
York University
Toronto, Ontario, Canada
komal253@yorku.ca

Zakeya Namrud*
York University
Toronto, Ontario, Canada
zakeya10@yorku.ca

Marin Litoiu
York University
Toronto, Ontario, Canada
mlitoiu@yorku.ca

Larisa Shwartz
IBM T. J. Watson Research Center
Yorktown Heights, New York, USA
lshwart@us.ibm.com

Ian Watts
IBM Canada Lab
Markham, Ontario, Canada
ifwatts@ca.ibm.com

## ABSTRACT

In the evolving landscape of software development and system operations, the demand for automating traditionally manual tasks has surged. Continuous operation and minimal downtimes highlight the need for automated detection and remediation of runtime anomalies. Ansible, known for its scalable features, including high-level abstraction and modularity, stands out as a reliable solution for managing complex systems securely. The challenge lies in creating an on-the-spot Ansible solution for dynamic auto-remediation, requiring a substantial dataset for in-context tuning of large language models (LLMs). Our research introduces KubePlaybook, a curated dataset with 130 natural language prompts for generating automation-focused remediation code scripts. After rigorous manual testing, the generated code achieved an impressive 98.86% accuracy rate, affirming the solution's reliability and performance in addressing dynamic auto-remediation complexities.

## CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; **Natural language processing**; • **Information systems** → **Information systems applications**.

## KEYWORDS

Kubernetes, Ansible Playbook, LLMs, GPT-4, Auto-remediation, Microservices.

**ACM Reference Format:**
Komal Sarda, Zakeya Namrud, Marin Litoiu, Larisa Shwartz, and Ian Watts. 2024. KubePlaybook: A Repository of Ansible Playbooks for Kubernetes Auto-Remediation with LLMs. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '24 ), May 7–11, 2024, London, United Kingdom.*https://doi.org/10.1145/3629527.3653665

*Both authors contributed equally to this work.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE Companion '24 , May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3653665

## 1 INTRODUCTION

In the dynamic software development landscape, the adoption of microservices has transformed scalability, flexibility, and agility [33, 39]. Kubernetes (K8s) [22], a key orchestration tool, plays a crucial role in managing microservices at scale, providing features like auto-scaling and self-healing [3]. For smaller production settings and local environments, MicroK8s, a lightweight K8s distribution, proves valuable [20]. It includes all essential components of a full K8s distribution, such as the API server, kubelet, and kubectl [21]. Kubectl, a vital command-line tool, simplifies the management and interaction with K8s clusters, enabling users to deploy applications, scale resources, create pods, and manage various K8s objects. Despite advancements in autonomic and adaptive computing [18, 35], many cloud services and applications still encounter failures, necessitating manual intervention. Additionally, the decentralized nature of microservices introduces complexity in detecting and resolving root-cause incidents [2, 16].

In microservices environments, automation plays a pivotal role in addressing complex issues swiftly [19]. AI-driven approaches have been integrated into IT operations (AIOps) particularly in self-healing processes using data-driven AI for automating incident life cycles [43]. However, challenges persist in manually creating remediation scripts, often relying on poorly organized troubleshooting guides [17]. For on-call engineers (OCEs) dealing with diverse anomalies across numerous services, the lack of organized guides can be time-consuming [2]. Anomalies, presenting differently or sharing traits across services, along with unique configuration settings, require meticulous attention. Minor script errors can lead to significant discrepancies, emphasizing the need for effective remediation scripts to expedite incident mitigation [18].

In response to these challenges, some researchers have turned to leveraging pre-trained Large Language Models (LLMs) to automatically generate remediation scripts based on identified root causes, a methodology successfully applied in various use cases [4, 5, 40]. Notable LLMs like CodeBert [7], Codex [5], LLaMa [38], GPT-Neo, GPT-NeoX [42], and GPT-4 [32] demonstrate promise for code generation tasks. While these techniques have been extensively applied in general-purpose programming languages, their adoption in IT domain-specific languages, particularly YAML, has received less attention. YAML files play a crucial role in defining and configuring key aspects of IT infrastructure [30]. In

specific IT domains, such as those utilizing Ansible-YAML [12] to manage infrastructure, the integration of LLMs can streamline incident response. Companies leverage Ansible playbooks in conjunction with K8s to design intelligent, automated responses to root cause alerts, reducing the burden of routine firefighting tasks on on-call engineers (OCEs).

While progress has been made in leveraging LLMs to generate Ansible playbooks, the focus has predominantly been on enhancing productivity for existing users, with an emphasis on code completion rather than the creation of entirely new code. A critical requirement emerges for specialized Ansible playbook generation LLMs tailored for auto-remediation, functioning as an AI assistant for OCEs [19]. To address challenges related to the cost and maintenance of traditional LLMs, researchers are turning to the few-shot learning capabilities of LLMs [17, 37]. This approach enables incident-specific code generation with minimal examples, eliminating the need for extensive parameter tuning. However, the effectiveness of these models is hindered by the lack of open-source, high-quality prompts and playbook corpora [31]. To bridge this gap, we aim to create the KubePlaybook dataset, dedicated to Ansible playbooks in the context of IT automation and anomaly resolution within cloud-native environments. This dataset is crucial for enhancing the few-shot learning capabilities of LLMs, enabling them to autonomously generate more Ansible Playbooks for auto-remediation throughout the incident life cycle. Addressing these challenges brings us closer to realizing a fully autonomous AIOps environment.

***Contribution:*** This paper introduces KubePlaybook, publicly accessible through a GitHub repository[1], a dataset featuring 130 Ansible playbooks accompanied by natural language (NL) prompts designed for code generation. NL prompts, serving as queries or descriptions, instruct LLMs to generate task-specific code. While the details about LLMs and prompts are not extensively covered due to page limitations, each NL prompt in KubePlaybook describes a root cause along with the operator's query input. Our evaluation process meticulously assesses the effectiveness of NL prompts in generating appropriate Ansible playbooks. Following this, we evaluate the functionality of each playbook by applying them to a sample microservices application to ensure their efficacy. The paper is structured as follows: Section 2 details dataset collection, generation, and description. Section 3 presents an experimental evaluation and results discussion. Subsequently, Section 4 deliberates on our work, highlighting challenges. Section 5 reviews relevant literature, and Section 6 provides conclusion and outlines future research directions.

## 2 KUBEPLAYBOOK FRAMEWORK

### 2.1 Data Collection & Generation

The development of the KubePlaybook repository employs a structured process as shown in Figure 1. We initially collect K8s Ansible playbooks from GitHub [9] and Galaxy [8], along with kubectl shell commands and real-time faults. Leveraging Ansible for system management, which is more scalable than traditional shell commands [14, 27], addresses automation challenges. To automate Ansible playbook creation for kubectl commands and faults, a playbooks generation approach using GPT-4 is adopted. An incident remediation dataset is built using root cause alerts and operator input as prompts. Few-shot learning on GPT-4 is applied to streamline the process, following a method adapted by many researchers [17, 37]. Initiating with 10 random kubectl commands, each is transformed into an Ansible playbook using purpose-specific prompts. The generated playbook, serving as the auto-remediation script, undergoes testing and manual examination. Adjustments are made to prompts until the desired Ansible-based remediation playbooks is obtained. A robust prompt template is integrated into KubePlaybook along with corresponding playbooks, and the few-shot learning dataset is updated for deployment with GPT-4. This iterative process is consistently applied to the initial 10 commands, tuning the GPT-4 model with newly generated samples. Careful adjustments are made to both prompts and playbooks, ensuring the creation of precise and context-sensitive playbooks tailored for incident resolution. Having generated and validated 10 structured prompt templates for auto-remediation in microservices, the same prompt structure is applied to the remaining 120 samples. The tuned GPT-4 model generates playbooks for the entire dataset, ensuring a efficient approach to incident resolution in the microservices environment.

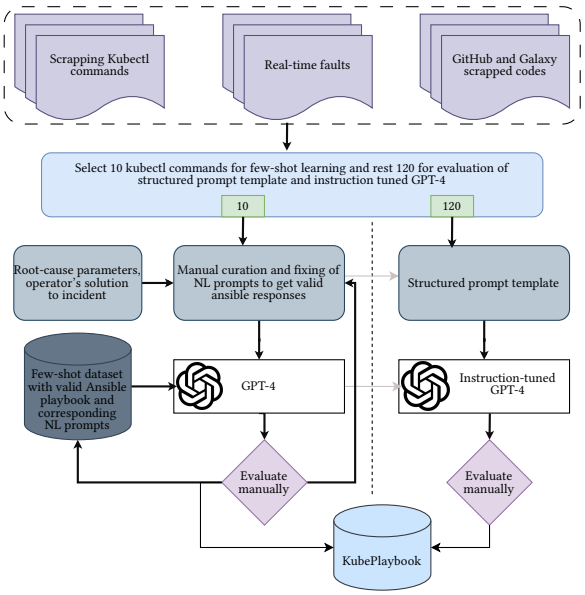



**Figure 1: Overview of Building the Dataset**

### 2.2 Composition of NL Prompts

To translate Kubectl commands and address real-time faults into valid Ansible playbook, it is crucial to construct well-defined prompts that provide detailed information for the AI model. This process involves crafting precise instructions to guide the model in automatically resolving issues within microservices architectures [30, 41]. Microservices architectures consist of multiple independent services, and the initial step involves specifying precise targets, such as host names, pod names, and deployment names, in prompts for LLM-driven remediation based

[1]https://github.com/K8sPlayBook/KubePlaybook

on root-cause alerts. Various anomaly detection models [34] and multimodal root-cause models [23, 24] are available to identify this information. This approach enables playbook to focus on specific services, minimizing disruptions to others. The next step is to define the desired automation actions, such as adjusting resources, restarting services, or implementing fixes for specific events. This action instructs the LLMs to generate Ansible playbook based on root-cause actions. This information can be derived directly from a root-cause mitigation recommendation model [2] or from operators. An example of our structured prompt and Ansible code is depicted in Figure 2. We selected smallest possible playbook for the example. This prompt incorporates placeholders like `<Host_Name>`, `<Name_space>`, and `<Deployment_name>` as indicators for inserting concrete values or variables, streamlining the management of deployments within a microservices framework. Additionally, the provided prompt serves as a guide for those involved in developing prompt templates using LLMs, demonstrating how to structure prompts to facilitate the automated generation of Ansible playbooks for addressing issues within microservices architectures. For practical application, placeholders should be replaced with actual values, and the resulting Ansible playbook must be customized to meet the unique requirements of the specific microservices environment and deployment details.

Write an Ansible playbook following these instructions:1) *Use kubectl command to get 'my-job' job info* for the deployment **'web'**, namespace **'Robot-shop'** and host **'k8s_nodes'** and *assert that **'my-job'** is ready*. 2) Use the command module to retrieve and display the deployment details in JSON format.

```
---
- name: K8s deployment update playbook
  hosts: k8s_nodes
  gather_facts: false
  tasks:
    - name: Get deployment list with "web"
      shell: |
        kubectl get deployments --all-namespaces | awk '{print $1,$2}' | grep web
      register: var_1
    - name: Split deployment name and namespace
      set_fact:
        deployment_name: "{{ var_1.stdout_lines[0].split()[1] }}"
        name_space: "{{ var_1.stdout_lines[0].split()[0] }}"
      when: "'web' in var_1.stdout"
    - name: Get job
      shell: |
        kubectl get jobs -n {{ name_space }} | grep my
      register: var_2
    - name: Display result in json format
      debug:
        var: var_2.stdout
```

**Figure 2: Sample of NL prompt & its corresponding Ansible playbook generated using GPT-4.**

## 2.3 KubePlaybook Description

The KubePlaybook dataset comprises NL prompts and Ansible playbooks, manifested as text and YAML files, respectively, aligning with targeted K8s commands. Our repository, outlined in Table 1, encompasses 130 Ansible playbooks categorized into three main classes, each featuring pairs of playbooks and corresponding NL prompts. According to Table 1, 62.3% of the playbooks focus on essential K8s commands, 19.2% are sourced from Ansible Galaxy and GitHub, and 18.5% are specifically designed for addressing chaos-engineered operational faults [26]. This categorization highlights the diverse nature of the playbooks in our dataset, covering aspects such as cluster management, real-time fault resolution, and external contributions. Our objective extends beyond constructing an Ansible playbook; we aim to extract meaningful NL prompts from each code. The repository mirrors the categorization format in directories for clarity and ease of navigation. As depicted in Figure 2, defining a placeholder in the prompt generates Ansible code that extracts target deployment details and applies the 'kubectl' command to the specified service. Notably, our dataset has no external package dependencies; it utilizes the latest versions of Ansible and Kubernetes.

**Table 1: Repository Overview Description**

| Categories | List of Ansible Playbooks & Prompts | |
|---|---|---|
| Generated Ansible playbook using LLMs Essential & common for (configuration & deployment) | Cluster Management (6), Daemonsets (6), Deployments (6), Events (5), Image name (1), Jobs (3), Logs (8), Namespaces (6), Nodes (11), Pods (10), ReplicaSets (3), Replication (2), Secrets (4), Service Accounts (3), Services (4), StatefulSet (3) | 62.3% |
| Ansible playbooks from Galaxy & GitHub | Collection of K8s tasks (25) | 19.2% |
| Real-time faults | DNS errors (1), DNS fault (1), Node I/O stress (1), Pod API latency (1), Overrides the header values of API requests (1), Node memory hog (1), Resources overload (2), Operational Error (4), Connection refused (6), Access denied (1), Login failure (1), Process crash (1), System stuck (1) | 18.5% |

# 3 EXPERIMENTAL SETUP & EVALUATION

## 3.1 Experiment Configuration

To guarantee the reliability and efficiency of Ansible playbooks generated using GPT-4, a thorough evaluation process precedes their production deployment. Our evaluation involved testing the efficacy of the dataset to adapt few-shot learning on LLMs. We utilized 10 samples for few-shot learning on GPT-4 and conducted evaluations on 120 samples. We utilized the GPT-4 model, specifically opting for the gpt-4-1106 [36]-preview version-recognized as the latest and most proficient model for code comprehension from OpenAI. Testing occurred on a t2.2xlarge EC2 instance with Ubuntu, installing Robot-shop [1] and QoTD [11]. We validated both syntactic correctness and functional soundness during this crucial phase, focusing on the effectiveness of automated deployment. Hyperparameter tuning for GPT-4 was tailored to our needs, utilizing maximum output length with 10 samples for few-shot data, optimizing within token count limits. After iterative experimentation, optimal hyperparameters-temperature [10] at 0.6 and Top-P [10] at (1.0)-were chosen to fine-tune the model's performance for precise YAML configurations from NL instructions. This meticulous testing and tuning ensure robust Ansible playbooks, ready for seamless deployment in real-world production environments.

## 3.2 Evaluation Methodology

We performed manual testing on each Ansible playbook on our setup to ensure successful execution. The Average Correctness (AC)

metric was employed for evaluation, focusing on the precision of individual code blocks or sub-tasks within Ansible playbooks. These sub-tasks, crucial units in Ansible, encompass actions like package installations, service configurations, or file transfers, influencing the overall playbook accuracy. The AC metric is defined by Equation 1, evaluating the accuracy of generated Ansible playbooks. For each Ansible playbook code (*APC*), *AC* computes the accuracy per task by comparing the correctly executed tasks ($C$) to the total tasks ($n$) in that *APC*. The accuracy ratios of all tasks across all *APCs* are summed, providing an aggregate view of performance. The final AC value represents the average of these task accuracy ratios, obtained by dividing the sum by the total number of *APCs* (m). This nuanced approach offers insights into the quality of individual code blocks beyond a binary overall judgment.

$$AC = \frac{\sum_{j=1}^{m}(C_j/N_j * 100)}{m} * 100 \quad (1)$$

We refrained from utilizing other lexical metrics such as BLEU-4 [28] or semantic metrics like BERTScore[28] due to the unavailability of ground truth. Table 2 presents the performance metrics, highlighting GPT-4's exceptional accuracy in generating Ansible playbook. With a 98.86% accuracy rate for code pertaining to Kubectl commands. Additionally, when assessed against 24 real-time faults, GPT-4 achieved a 92.36% accuracy rate. This stands in stark contrast to a 60% accuracy rate for identical tasks performed by humans.

**Table 2: Evaluation of Ansible playbooks**

| Task Source | Accuracy |
|---|---|
| Kubectl command | 98.86% |
| Code was written by a human from GitHub | 60% |
| Real-time fault | 92.36% |

## 4 DISCUSSION & CHALLENGES

To foster operator trust in automated code generation for expediting incident resolution, a large, real-world dataset is crucial for fine-tuning models. Towards this, our dataset serves as a valuable starting point. We devised a specific pattern for constructing prompts using operator inputs, recognizing the potential variations across industries. The evaluation involved testing prompt construction and Ansible playbook generation on e-commerce microservice applications like robot-shop and QoTD using Kubectl. However, it's important to note that these results may not universally apply to different microservice applications and tools. Additionally, while the GPT-4 were instruction-tuned and evaluated to generate a specific dataset, variations in results and accuracy may occur for different datasets. Rigorous manual tweaking and testing were performed to ensure the reliability of each playbook on our setup.

## 5 RELATED WORK & APPLICATION

Benchmark datasets have become crucial for advancing applied AI research, particularly as the demand for evaluating models across diverse applications grows. Among the key contributions to this area, Hendrycks et al.[13] pioneered the assessment of large transformer language models in competitive programming with the introduction of the APPS dataset, featuring 10,000 coding competition problems. The CodeXGLUE [25] and CodeSearchNet [15] datasets further extend the toolset for researchers with tasks ranging from code summarization to code translation, and offering professional annotations for NL queries across several programming languages. However, Ansible was not considered. 20-MAD [6] is a dataset that links the commit and issue data of the Mozilla and Apache projects. Our work distinguishes itself by focusing on the generation of Ansible playbooks, a niche not covered by the aforementioned datasets that primarily utilize NL for code generation. Although the work of Ahmed et al.[2] is closely related through its use of LLMs for text generation, our emphasis remains on code generation. Moreover, unlike the Andromeda [29], which provides an overview of the Ansible Galaxy ecosystem, our dataset is specifically tailored for training LLMs with generated and scraped Ansible playbooks along with their associated prompts. To our knowledge, KubePlaybook is the starting point for K8s-based Ansible playbook benchmark dataset crafted using LLMs. Aimed at auto-remediation microservices, it sets a new precedent for employing LLM-generated datasets in practical applications. Table 3 outlines the distinctions between the current cutting-edge research initiatives and our methodology. This includes the employment of LLMs for the generation of the Ansible playbook, the automation of code scraping techniques, the detailed generation of report descriptions, and the careful crafting involved in prompt engineering.

**Table 3: Comparison between state-of-the-art research and our approach.**

| References | LLMs usage | Scraped repositories | Ansible playbook | NL prompts |
|---|---|---|---|---|
| Paper [2] | ✗ | ✓ | ✗ | ✓ |
| Paper [5] | ✗ | ✓ | ✗ | ✗ |
| Paper [6] | ✗ | ✓ | ✗ | ✗ |
| Paper [13] | ✓ | ✗ | ✗ | ✗ |
| Paper [15] | ✓ | ✓ | ✗ | ✗ |
| Paper [25] | ✓ | ✗ | ✗ | ✗ |
| Paper [29] | ✗ | ✓ | ✓ | ✗ |
| Our Dataset | ✓ | ✓ | ✓ | ✓ |

## 6 CONCLUSION & FUTURE DIRECTIONS

In conclusion, while the application of AI, especially LLMs, in automating IT operations holds promise for self-healing mechanisms, challenges persist in applying these advancements to IT-centric languages like YAML. The introduction of the KubePlaybook dataset, comprising 130 NL prompts and Ansible playbooks, represents a significant step towards addressing this gap. It facilitates the contextual learning of LLMs to generate Ansible-YAML scripts for automated remediation tasks, a pivotal development for advancing IT automation in cloud-native environments. It is poised to enhance incident response in the dynamic realm of IT operations. Future research may focus on augmenting playbook quality by incorporating more real-time fault data examples. The manual creation of corresponding NL prompts for each script, a meticulous and time-consuming process, motivates us to explore automatic prompt generation for auto-remediation tasks in microservices environments. Additionally, investigating the performance and adaptability of different LLMs for playbook generation presents a potential area of exploration.

# REFERENCES


[1] 1steveww et al. 2023. Robot Shop is a sample microservice application. Retrieved Mar 6, 2023 from https://github.com/instana/robot-shop

[2] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. *arXiv preprint arXiv:2301.03797* (2023).

[3] Meriem Azaiez and Walid Chainbi. 2016. A multi-agent system architecture for self-healing cloud infrastructure. In *Proceedings of the International Conference on Internet of things and Cloud Computing*. 1–6.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[6] Maëlick Claes and Mika V Mäntylä. 2020. 20-MAD: 20 years of issues and commits of Mozilla and Apache development. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 503–507.

[7] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[8] Ansible Galaxy. 2024. Ansible Galaxy. Retrieved Jan 30, 2024 from https://galaxy.ansible.com/ui/collections/

[9] GitHub. 2024. GitHub. Retrieved Jan 30, 2024 from https://github.com/

[10] Fabian Gloeckle, Baptiste Roziere, Amaury Hayat, and Gabriel Synnaeve. 2023. Temperature-scaled large language models for Lean proofstep prediction. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*.

[11] Red Hat. 2023. QoTD. Retrieved Oct 11, 2023 from https://github.com/redhat-developer-demos/qotd.git

[12] Red Hat. 2023. Red Hat Ansible Automation Platform. Retrieved Nov 27, 2023 from https://www.redhat.com/en/technologies/management/ansible

[13] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938* (2021).

[14] Eric Horton and Chris Parnin. 2022. Dozer: migrating shell commands to ansible modules via execution profiling and synthesis. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. 147–148.

[15] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).

[16] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, et al. 2020. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1410–1420.

[17] Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, et al. 2023. Xpert: Empowering Incident Management with Query Recommendations via Large Language Models. *arXiv preprint arXiv:2312.11988* (2023).

[18] Cornel Klein, Reiner Schmid, Christian Leuxner, Wassiou Sitou, and Bernd Spanfelner. 2008. A survey of context adaptation in autonomic computing. In *Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08)*. IEEE, 106–111.

[19] Sarda Komal, Namrud Zakeya, Rouf Raphael, Ahuja Harit, Rasolroveicy Mohammadreza, Litoiu Marin, Shwartz Larisa, and Watts Ian. 2023. ADARMA Auto-Detection and Auto-Remediation of Microservice Anomalies by Leveraging Large Language Models. In *Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering*. 200–205.

[20] Heiko Koziolek and Nafise Eskandani. 2023. Lightweight Kubernetes Distributions: A Performance Comparison of MicroK8s, k3s, k0s, and Microshift. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*. 17–29.

[21] Kubectl. 2024. Kubectl. Retrieved Jan 30, 2024 from https://kubernetes.io/docs/reference/kubectl/

[22] Kubernetes. 2024. Kubernetes. Retrieved Jan 30, 2024 from https://kubernetes.io/

[23] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 1–10.

[24] Fred Lin, Keyur Muzumdar, Nikolay Pavlovich Laptev, Mihai-Valentin Curelea, Seunghak Lee, and Sriram Sankar. 2020. Fast dimensional analysis for root cause investigation in a large-scale service environment. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2 (2020), 1–23.

[25] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664* (2021).

[26] Sehrish Malik, Moeen Ali Naqvi, and Leon Moonen. 2023. CHESS: A Framework for Evaluation of Self-adaptive Systems based on Chaos Engineering. *arXiv preprint arXiv:2303.07283* (2023).

[27] Pavel Masek, Martin Stusek, Jan Krejci, Krystof Zeman, Jiri Pokorny, and Marek Kudlacek. 2018. Unleashing full potential of ansible framework: University labs administration. In *2018 22nd conference of open innovations association (FRUCT)*. IEEE, 144–150.

[28] Nabor C Mendonça, Pooyan Jamshidi, David Garlan, and Claus Pahl. 2019. Developing self-adaptive microservice systems: Challenges and directions. *IEEE Software* 38, 2 (2019), 70–79.

[29] Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. 2021. Andromeda: A dataset of Ansible Galaxy roles and their evolution. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 580–584.

[30] Saurabh Pujar, Luca Buratti, Xiaojie Guo, Nicolas Dupuis, Burn Lewis, Sahil Suneja, Atin Sood, Ganesh Nalawade, Matt Jones, Alessandro Morari, et al. 2023. Automated Code generation for Information Technology Tasks in YAML through Large Language Models. *arXiv preprint arXiv:2305.02783* (2023).

[31] Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–7.

[32] Katharine Sanderson. 2023. GPT-4 is here: what scientists think. *Nature* 615, 7954 (2023), 773.

[33] Komal Sarda. 2023. Leveraging Large Language Models for Auto-remediation in Microservices Architecture. In *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE, 16–18.

[34] Jacopo Soldani and Antonio Brogi. 2022. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)* 55, 3 (2022), 1–39.

[35] Roy Sterritt, Manish Parashar, Huaglory Tianfield, and Rainer Unland. 2005. A concise introduction to autonomic computing. *Advanced engineering informatics* 19, 3 (2005), 181–187.

[36] Kaiming Tao, Zachary A Osman, Philip L Tzou, Soo-Yon Rhee, Vineet Ahluwalia, and Robert W Shafer. 2024. GPT-4 Performance on Querying Scientific Publications: Reproducibility, Accuracy, and Impact of an Instruction Sheet. (2024).

[37] Catherine Tony, Markus Mutas, Nicolás E Díaz Ferreyra, and Riccardo Scandariato. 2023. LLMSecEval: A Dataset of Natural Language Prompts for Security Evaluations. *arXiv preprint arXiv:2303.09384* (2023).

[38] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[39] Hulya Vural, Murat Koyuncu, and Sinem Guney. 2017. A systematic literature review on microservices. In *Computational Science and Its Applications–ICCSA 2017: 17th International Conference, Trieste, Italy, July 3-6, 2017, Proceedings, Part VI 17*. Springer, 203–217.

[40] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).

[41] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382* (2023).

[42] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 1–10.

[43] Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. 2016. Early detection of configuration errors to reduce failure damage. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 619–634.

# Efficient Unsupervised Latency Culprit Ranking in Distributed Traces with GNN and Critical Path Analysis

Mahsa Panahandeh
Electrical and Computer Engineering department,
University of Alberta
Edmonton, Alberta, Canada
panahand@ualberta.ca

Naser Ezzati-Jivan
Department of Computer Science, Brock University
St. Catharines, Ontario, Canada
nezzatijivan@brocku.ca

Abdelwahab Hamou-Lhadj
Department of Electrical and Computer Engineering,
Concordia University
Montreal, Quebec, Canada
wahab.hamou-lhadj@concordia.ca

James Miller
Department of Electrical and Computer Engineering,
University of Alberta, Canada
Edmonton, Alberta, Canada
jimm@ualberta.ca

## ABSTRACT

Microservices offer the benefits of scalable flexibility and rapid deployment, making them a preferred architecture in today's IT industry. However, their dynamic nature increases their susceptibility to failures, highlighting the need for effective troubleshooting strategies. Current methods for pinpointing issues in microservices often depend on impractical supervision or rest on unrealistic assumptions. We propose a novel approach using graph unsupervised neural networks and critical path analysis to address these limitations. Our experiments on four open-source microservice benchmarks show significant results, with top-1 accuracy ranging from 86.4% to 96%, over 6% enhancement compared to existing methods. Moreover, our approach reduces training time by 5.6 times compared to similar works on the same datasets.

## CCS CONCEPTS

• **Software and its engineering → Software defect analysis**; • **Computer systems organization → Reliability**.

## KEYWORDS

Culprit identification, Graph neural Network, Critical path analysis, Distributed traces, FIRM dataset

**ACM Reference Format:**
Mahsa Panahandeh, Naser Ezzati-Jivan, Abdelwahab Hamou-Lhadj, and James Miller. 2024. Efficient Unsupervised Latency Culprit Ranking in Distributed Traces with GNN and Critical Path Analysis . In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3651841

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651841

## 1 INTRODUCTION

Microservice architecture is becoming increasingly popular among various systems architectures due to its fast delivery, scalability, and independence. However, managing quality in microservice applications remains a significant challenge [5, 6, 10]. This challenge involves the need to set suitable alert thresholds and filters to alert developers to issues promptly, without overwhelming them with extraneous data [6]. Hassan et al. [5] and Jamshidi et al.[6] advocate that machine learning could significantly mitigate these challenges. As a result, numerous trace-based methods employing machine learning techniques have been developed to quickly identify and address issues as they emerge [8, 10, 12]. However, these methods face challenges, primarily due to idealistic assumptions or relying on supervision methods, reducing their effectiveness and practical applicability [10].

Numerous studies [8, 14, 16, 19] have employed supervised machine learning models for anomaly detection and predicting root causes in microservices. These approaches heavily depend on labelled datasets, which can be costly to acquire [7]. Furthermore, they require comprehensive coverage of all possible fault types to accurately differentiate between anomaly propagation paths. Yet, some of these studies do not incorporate request types or fault types in their approach, and they may also lack this information, e.g., [16].

Although certain studies [11, 12, 16] overlook the variability in anomaly propagation patterns, other research works [10, 14] emphasise the crucial importance of considering this variability to enhance anomaly detection and analysis. For example, FIRM [14] emphasizes how latency anomalies may propagate differently across scenarios, even for identical request types. To investigate variations of anomaly propagation paths, FIRM [14] proposes studying critical paths in request executions. It employs a supervised classification approach, which still makes it rely on labelled datasets including both normal and abnormal requests of different execution paths. Moreover, FIRM [14] assumes that the longest service on a critical path contributes most to the total latency. However, our experiments show that this assumption does not always hold because individual services can show a wide variance in latency, even under normal conditions.

Addressing the need for distinguishing between anomaly propagation paths as well as breaking the need for having a labelled dataset, Li et al.[10] propose a trace analysis approach based on studying historical data of service invocations. Although Li et al. study[10] outperforms other unsupervised methods, it may not be optimal for microservice systems handling extensive requests involving numerous services. The issue arises from their method requiring feature selection for each service invocation within a real-time window, which can result in significant time consumption. Additionally, the effectiveness of Li et al.'s approach[10] might be compromised in scenarios with a scarce number of normal or abnormal traces, or when the test window includes a limited range of request types.

To overcome these limitations, we developed an unsupervised method that integrates the analysis of critical paths for enhanced culprit prioritization. Our approach starts with establishing a baseline model using the application of graph neural networks (GNN), which learns the expected latency distributions across all services and their interdependencies, effectively mirroring the system's anticipated behaviour. This model is then used for anomaly detection at the granularity of individual requests. To narrow down the context of latency propagation and identify culprits, we study abnormal requests against normal requests, sharing the same critical path. Our method involves clustering historical requests based on their critical paths, thereby creating distinct profiles for each path. Within these profiles, we detail vectors for each service that capture the service's distribution within the critical path. By comparing the observed latency of services in abnormal paths to their standard distributions outlined in the profiles, we effectively isolate and identify the culprits. Services exhibiting greater deviation from their expected distribution are ranked as more suspicious culprits.

To enhance the efficiency of our method compared to existing approaches, We design our GNN model with the assumption that the service invocations graph is static. By this assumption, we effectively reduce the complexity associated with dynamic network models, leading to increased efficiency. To prevent missing any update in service invocations, we propose periodic evolutionary updates as an alternative to ensure the model stays in sync with any changes. Additionally, we integrate a feature sampling technique from services across various layers in our GNN model. This strategy ensures the capture of critical information from services within the GNN, maintaining the model's scalability and efficiency, particularly when dealing with extensive service invocation or service dependency graphs.

Our experiments conducted on four benchmark systems in different sizes and complexity demonstrate that our approach has an accuracy of 86.4%-96% and outperforms similar methods [16] by over 6% while also enhancing the training time by a factor of 5.6. Our main contributions can be summarized as follows:

- Proposing an unsupervised GNN model, eliminating the requirement for labelled datasets which can be expensive to obtain in practice.
- Integration of our latency detection methodology with critical path analysis, refining the focus on potential culprits and enhancing the efficiency of culprit prediction.
- Refinement of the GNN model to tackle scalability challenges inherent in large service invocations or service dependency graphs, as well as improving time efficiency during both training and testing phases.

## 2 BACKGROUND SUMMARY

**Critical Path Analysis** involves identifying the longest duration path in a request's journey through a distributed system, which can be complex due to mixed synchronous and asynchronous communications. Distributed tracing helps trace these paths by collecting detailed request flow data [1, 14].

**Culprit Identification** focuses on pinpointing the service or component causing latency anomalies. While 'culprit' refers to the direct cause of performance slowdowns, 'root cause' delves into the underlying issue. This work prioritizes culprit ranking in abnormal request paths as a step towards root cause analysis [9].

## 3 METHOD DESIGN

Our approach introduces an end-to-end latency anomaly detection and culprit ranking framework, which comprises three primary phases: I) Data Preparation, II) Anomaly Detection, and III) Culprit Ranking, as detailed in the subsequent subsections.

### 3.1 Data Preparation

Our approach takes two main inputs: historical data representing the expected behaviour of the system and interdependencies between services. This data is collected from telemetry data provided by any distributed traces monitoring tool. We ensure our data encompasses a wide range of requests, each with a unique execution path. Each request is collected as a trace that can be defined by a feature vector that details the latency of services (spans) involved in the request:

$$V_{R_0} = [l_1, l_2, ..., l_n], \tag{1}$$

Here, n is the number of services in request $R_0$ and $l$ is the latency value for each service. The historical data comprises numerous such feature vectors. To ensure consistency across the data, we normalize all latency values. Then, we determine the critical path for each request using the method outlined in the literature [14], and accordingly cluster the data based on critical paths. For each cluster, we create a profile that encapsulates the distribution of latencies, including mean and standard deviation, for each service within the critical path. Such a standardized approach allows for a comprehensive comparison of service performance across various service paths.

Figure 1 presents latency distributions for a service (10-contact) in a ticket-booking benchmark [20]. On the left, it shows the latency distribution across all requests in historical data, while on the right, latency distributions across different critical paths are shown. The distribution of data across clusters facilitates a more accurate discernment of patterns. Such clarity is especially beneficial for data lacking obvious service relationships or evident request clustering patterns, as observed in our experiments about the dataset used in this study, where no clustering algorithm, such as k-means or hierarchical clustering, yields meaningful clusters. This limitation makes some existing methods [13, 18] inappropriate for such data.

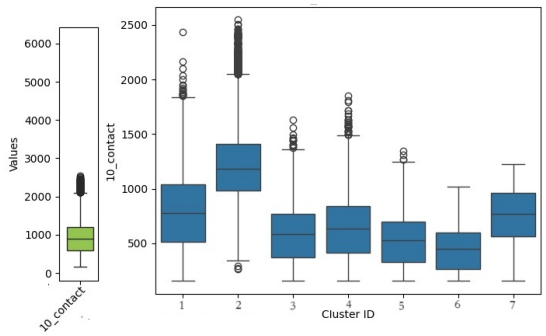



**Figure 1: The latency distribution of a service across all execution paths (left) compared to within critical paths (right).**

The second input that our approach takes is interdependencies between services such as service invocations, as we believe that these interdependencies primarily govern the propagation of behaviour across the services. This data can be inferred from dependency graphs provided by distributed trace tools or by studying the microservice system itself.

## 3.2 Anomaly detection

Utilizing the advanced pattern recognition capabilities of deep graph neural networks (GNNs) for structured data analysis, we use a GNN model to establish a baseline for anomaly detection. Our model is constructed based on interdependencies patterns and distribution of latencies, collected from historical data. First, we generate a directed acyclic graph G= (V,E) from service interdependencies, where V represents the set of nodes corresponding to services and E represents the set of edges representing service invocations. The direction of each edge specifies the propagation path of behaviour through services. Each node is associated with a feature array that encapsulates the corresponding latency values from all request feature vectors ($V_R$ vectors) in historical data. To enhance the efficiency of our model compared to existing works [8, 16], we consider the following considerations.

Firstly, we initially assume a static structure for the directed acyclic graph to enhance the GNN's training efficiency and scalability. This assumption is pivotal for managing large call graphs and complex service interaction patterns. To adapt to changes in service invocations, we introduce periodic model updates, allowing our system to reflect changes over time without compromising the initial efficiency gains.

Secondly, we implement batch processing, enabling simultaneous analysis of multiple graphs, each with unique structures and sizes. This diversity is essential in training our model to recognize a wide array of connectivity patterns. Our adaptation employs a neighbourhood sampling strategy, where a fixed-size subset of a node's neighbours is selected for feature aggregation. This method addresses challenges related to variable connectivity and scales effectively for large graphs. Each layer aggregates information from a node's immediate neighbours and subsequent layers aggregate information at increasingly larger distances from the target node. This iterative process ensures that even with sampling, information from the broader neighbourhood can influence the node's representation, albeit indirectly. We mitigate potential data loss from sampling by designing our model to aggregate features across layers. Unlike traditional methods that depend on separate embeddings for each node to produce features, our design consolidates feature generation across the network.

We adopted the GraphSAGE model [4], to construct our model according to these considerations. Our model comprises two graph convolutional layers. The initial layer transforms the features of a node and its neighbours, aggregated through a mean function, into a dimensional hidden space, for abstract representation. The second layer then projects these hidden representations back to the original feature dimensions. This unsupervised learning architecture is adept at encoding and decoding node features, capturing both structural and feature-based graph information to generalize well to unseen nodes or entirely new graphs.

The model's operation during each message-passing iteration is mathematically described as follows:

$$H_v^{(l+1)} = \sigma\left(W^{(l)} \cdot \text{MEAN}\left(\{H_v^{(l)}\} \cup \{H_u^{(l)}, \forall u \in \mathcal{N}(v)\}\right)\right) \quad (2)$$

where $H_v^{(l)}$ is the feature vector of node $v$ at layer $l$, $W^{(l)}$ is the weight matrix for layer $l$, $\sigma$ denotes a non-linear activation function (e.g., ReLU), and $\mathcal{N}(v)$ includes the neighbours of node $v$. Our optimization goal is to minimize the mean squared error (MSE) between the original node features ($X$) and their reconstructed counterparts ($\hat{X}$) after processing through the GraphSAGE layers. The optimization goal is succinctly captured by:

$$\text{Minimize } \mathcal{L} = \text{MSE}(X, \hat{X}) \quad (3)$$

Here, $X$ represents the model's input (the original node features), and $\hat{X}$ denotes the output (the reconstructed node features).

To detect anomalies using the trained modes, we convert a test trace into a format that aligns with the request feature vector structure previously outlined. We use our model ($f$) along with adjacency information encapsulated in the graph ($E$) to reconstruct the test request feature vector $X$ as $\hat{X} = f(X, E)$. Anomalies are identified by comparing the loss between the original and reconstructed vectors against a threshold, i.e., $MSE(X, \hat{X}) > threshold$. In this paper, we set the threshold at 0.1.

## 3.3 Culprit Ranking

After detecting an anomaly, we proceed with the Culprit Ranking step by comparing the abnormal test request's feature vector with similar requests from historical data. Our experiments indicate that narrowing these comparisons to critical paths is more efficient. Critical paths encompass services that drive the latency issue in a request, making them a priority for identifying the culprit.

We first, compute the critical path within the test request. Then, we match it with the same critical path cluster in historical data. From this match, we extract the cluster profile that provides detailed insights into the distribution of each service along the critical path. For each service within the test critical path, we hypothesize it is a potential culprit. Then, we use the service distribution details extracted from the critical path profile to choose an appropriate statistical simulation technique, as advised by Forbes [2]. Through

parameter estimation, we generate a new latency value for the service that aligns with its distribution. This value replaces the original one in the test vector, and anomaly detection is performed on this manipulated request vector. If the model identifies the manipulated vector as normal, our hypothesis is validated, and the service is added to the list of culprits; otherwise, it's discarded. Culprits are subsequently ranked based on their deviation from their respective normal distributions.

During the validation of our model, we discovered that checking services individually as potential culprits aids in identifying indirect anomaly propagation paths. We call frequently affected services by a true root cause, even when there are no direct invocations between them, "indirect dependency". Figure 2-**A**, shows direct dependencies, i.e., service invocations, as well as indirect dependencies for the hotel reservation benchmark system. This representation assists in understanding anomaly propagation paths and can be integrated into our model to improve accuracy by accommodating unknown service dependencies.

## 4 EXPERIMENTS AND EVALUATION

We evaluate our approach on a dataset including four microservice benchmarks provided by Qiu et al.[15] as part of the FIRM research project[14]. These benchmarks cover social networking, hotel reservations, media services, and ticket booking systems. The dataset includes traces representing normal system behaviour, with latency data for all services, which we treat as our historical dataset. Additionally, there are labelled files for each benchmark system, containing traces collected during anomalies in labelled services, treated as test requests for evaluating our approach. The dataset also includes covered execution paths of collected requests, illustrating the interdependency patterns within each benchmark system.

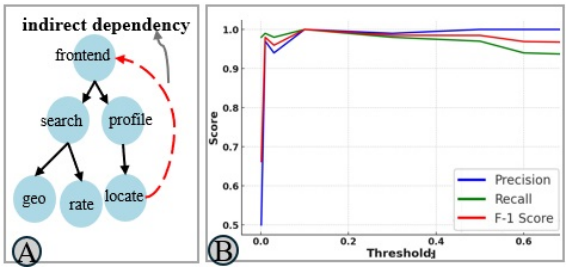



**Figure 2: A: Direct and indirect anomaly propagation paths, B: Confusion metrics for benchmark systems**

The performed exploratory data analysis (EDA) on the data, combined with in-depth studies and collaborative consultations with the FIRM authors, reveals that each sample in this dataset isn't simply a raw trace, but rather an aggregation of collected requests spanning the system's execution paths within a 1-minute time window. Each sample provides all services' average latency time for handling various requests within this window. Despite being aggregated from numerous requests, there is consistency in workload and covered execution paths for each sample. Therefore, each sample can be treated as a trace covering all services and execution paths,

| Benchmarks | ACC | Top-1 | Top-3 | Top-5 |
|---|---|---|---|---|
| Social-network (MSE=0.005) | 87% | 83% | 86.2% | 87% |
| hotel-reservation (MSE=0.001) | 95.4% | 94.7% | 95.4% | 95.4% |
| media-service (MSE=0.007) | 86.4% | 85% | 86.4% | 86.4% |
| ticket-booking (MSE=0.001) | 96% | 94.3% | 96% | 96% |

**Table 1: Culprit localization results for different benchmarks**

as commonly done in the literature [16, 17]. Consequently, within this pre-processed dataset, the critical path of each sample (trace) is a system critical path and can be computed as the execution path with the maximum latency among all other execution paths.

In our study, we constructed datasets comprising 60% training data, with an additional 20% set aside for both validation and testing of our model. To assess the efficacy of the culprit ranking, we employed the same sampling rate used in prior literature [16], selecting 20% of labelled files.

### 4.1 Results

We conduct evaluations for anomaly detection and culprit ranking separately. Figure 2-**B**, shows averaged results of precision, recall, and F-1 score metrics [3] across all benchmarks. This visualization illustrates the performance of our model in classifying anomalies for various classification threshold values. Based on our findings, we observed that setting the anomaly classification threshold lower than 0.1 results in a decrease in precision. Conversely, for larger threshold values, there is a decrease in recall while precision remains stable. Therefore, we opted to set the threshold at 0.1 to strike a balance in the performance of our model.

To evaluate the effectiveness of our approach in identifying the true culprit, we employ two key metrics: the overall percentage of accurately identified true culprits (referred to as "ACC") and the Top-k metric, which measures the likelihood of ranking the true culprit within the top k locations among all ranked identified culprits. Table 1 displays the performance results for each benchmark. The numerical values provided alongside the name of each benchmark indicate the average loss value incurred by our trained model for that particular benchmark. Our experiments across four case studies indicate that the true culprit is accurately identified in 86.4% to 96% of cases, and is also ranked within the first five culprits in 83% to 94.7% of cases. Our investigation unveiled that having a balanced representation of requests related to various critical paths in the historical data significantly enhances the ability to distinguish between service distributions of the test and historical critical paths which leads to higher accuracy in our analysis.

In comparison with existing works, we create a dataset with a similar train-test split ratio used by B-MEG [16], that proposes a supervised GNN for the same dataset. We conducted multiple iterations of experiments and reported the average result. Our experiments yielded accuracy improvements of approximately 3-8%, along with a reduction in time complexity to over one-fifth of the model training. Due to the need for adjustments in B-MEG, we were unable to compare execution times for processing test requests in the current paper. However, we compared two variations of our approach: v1 solely utilises the main request path in culprit identification, and v2 incorporates the critical paths to narrow down the

test request context. Our results demonstrated that v2 significantly improves processing time for identifying the culprits in different benchmarks from 1.5-13 seconds in v1 to 0.8-8.3 seconds, marking a 58.33% enhancement on average. Figure 3 summarises the average performance of two versions of our approach against Somashekar et al. work [16].

Despite improvements in v2 of our approach, occasional slight accuracy decreases were observed, although accuracy increased for ticket-booking and social-network benchmarks. Further investigation of our dataset revealed that the stated computed critical path method for this dataset may overshadow the true culprit. This happens for systems with sparse connectivity and shorter request paths since they have limited service diversity in different execution paths. Therefore, the computed critical path can be affected if the true culprit exhibits a relatively small latency distribution range compared to other services.

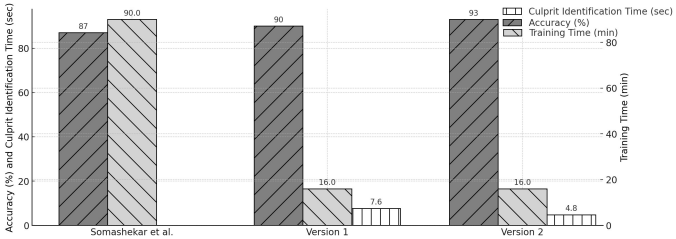



**Figure 3: Comparison of two different versions of our approach with B-MEG**

## 5 CONCLUSION

In this paper, we propose an efficient unsupervised GNN model integrated with critical path analysis for culprit ranking in microservice architectures. Our approach eliminates the need for labelled datasets, incorporates critical path analysis for efficient culprit detection, and introduces enhancements for the scalability and speed of the GNN model. Conducted experiments on four benchmarks our model achieves an accuracy in culprit identification ranging from 86.4% to 96%, representing a notable improvement of over 6% compared to existing methods. Furthermore, our approach reduces training time by 5.6 times compared to similar works. Additionally, leveraging critical path analysis results in a 58.33% enhancement in culprit identification speed. Looking ahead, we aim to refine our methodology through additional experiments aimed at broadening its applicability and improving detection capabilities.
The scripts of our model are accessible via the following link: https://anonymous.4open.science/r/ICPE2024-4281/v2.py.

## REFERENCES

[1] Brian Eaton, Jeff Stewart, Jon Tedesco, and N. Cihan Tas. 2022. Distributed Latency Profiling through Critical Path Tracing: CPT can provide actionable and precise latency analysis. *Queue* 20, 1 (mar 2022), 40–79. https://doi.org/10.1145/3526967

[2] Catherine Forbes, Merran Evans, Nicholas Hastings, and Brian Peacock. 2011. *Statistical distributions.* John Wiley & Sons.

[3] Cyril Goutte and Eric Gaussier. 2005. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In *European conference on information retrieval.* Springer, 345–359.

[4] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[5] Ahmed Hassan and Tamilselvan Arjunan. 2024. A Comparative Study of Deep Neural Networks and Support Vector Machines for Unsupervised Anomaly Detection in Cloud Computing Environments. *Quarterly Journal of Emerging Technologies and Innovations* 9, 1 (2024), 15–24.

[6] Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The journey so far and challenges ahead. *IEEE Software* 35, 3 (2018), 24–35.

[7] Iman Kohyarnejadfard, Daniel Aloise, Michel R Dagenais, and Mahsa Shakeri. 2021. A framework for detecting system performance anomalies using tracing data analysis. *Entropy* 23, 8 (2021), 1011.

[8] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-source Data. *arXiv preprint arXiv:2302.05092* (2023).

[9] Richard Li, Min Du, Zheng Wang, Hyunseok Chang, Sarit Mukherjee, and Eric Eide. 2022. LongTale: Toward Automatic Performance Anomaly Explanation in Microservices. In *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering.* 5–16.

[10] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS).* IEEE, 1–10.

[11] JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16.* Springer, 3–20.

[12] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, et al. 2020. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE).* IEEE, 48–58.

[13] Mahsa Panahandeh, Abdelwahab Hamou-Lhadj, Mohammad Hamdaqa, and James Miller. 2024. ServiceAnomaly: An anomaly detection approach in microservices using distributed traces and profiling metrics. *Journal of Systems and Software* 209 (2024), 111917.

[14] Haoran Qiu, Subho S Banerjee, Saurabh Jha, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. 2020. {FIRM}: An intelligent fine-grained resource management framework for {SLO-Oriented} microservices. In *14th USENIX symposium on operating systems design and implementation (OSDI 20).* 805–825.

[15] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2020. Pre-processed Tracing Data for Popular Microservice Benchmarks. https://doi.org/10.13012/B2IDB-6738796_V1

[16] Gagan Somashekar, Anurag Dutt, Rohith Vaddavalli, Sai Bhargav Varanasi, and Anshul Gandhi. 2022. B-MEG: Bottlenecked-microservices extraction using graph neural networks. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering.* 7–11.

[17] G Somashekar, A Suresh, S Tyagi, V Dhyani, K Donkada, A Pradhan, and A Gandhi. 2022. Reducing the Tail Latency of Microservices Applications via Optimal Configuration Tuning. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS).* IEEE, 111–120.

[18] Guangba Yu, Zicheng Huang, and Pengfei Chen. 2023. TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. *Journal of Software: Evolution and Process* 35, 10 (2023), e2413.

[19] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 683–694.

[20] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chenjie Xu, Chao Ji, and Wenyun Zhao. 2018. Benchmarking microservice systems for software engineering research. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings.* 323–324.

# Network Analysis of Microservices: A Case Study on Alibaba Production Clusters

Ghazal Khodabandeh
Brock University
St. Catharines, Ontario, Canada
gkhodobandeh@brocku.ca

Alireza Ezaz
Brock University
St. Catharines, Ontario, Canada
sezaz@brocku.ca

Naser Ezzati-Jivan
Brock University
St. Catharines, Ontario, Canada
nezzatijivan@brocku.ca

## ABSTRACT

Having an observation of the microservices connections complexities within a service is essential for system management and optimization. In this study, we analyzed a dataset of microservice traces from Alibaba's production clusters, segmenting call graphs based on services. Using a community detection model, we uncovered the connections between microservices within each service by finding collaborative patterns and dependencies. Expanding our analysis, we identified similarities among service graphs using clustering techniques. These findings provide detailed insights for system optimization and decision-making, offering a roadmap for using the constructed runtime microservices network behavior to improve overall system efficiency and performance.

## CCS CONCEPTS

• **Computing methodologies** → **Classification and regression trees**; • **Applied computing** → **Service-oriented architectures**.

## KEYWORDS

Service, Microservice, Community Detection, Graph.

**ACM Reference Format:**
Ghazal Khodabandeh, Alireza Ezaz, and Naser Ezzati-Jivan. 2024. Network Analysis of Microservices: A Case Study on Alibaba Production Clusters . In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3651842

## 1 INTRODUCTION

On the topic of microservices architecture, challenges come from the relationships microservices and services have with each other. A service is a self-contained unit of functionality within a software system. Microservices inside a service have relations with each other to make that service function. Also, services can communicate with each other through well-defined interfaces. This will cause complexities in deployment, scaling, and monitoring in a multi-service framework. Each microservice has distinct functions for user handling, business logic, and backend operations. This format of application is different and more complicated than the monolithic application architecture[2]. To address these challenges we should have a deep and wide insight into this kind of system.

Despite extensive research on microservices [2, 7, 11], the application of social network analysis in large-scale industrial settings is uncommon. This oversight is significant in environments needing detailed analysis of microservices interactions. Our study addresses this gap by exploring these interactions within a major industrial context, aiming to enhance system performance and reliability through novel insights and methodologies.

To highlight the dynamics within microservices relationships, our study relies on a dataset containing over 260 million records of call requests across twenty thousand distinct microservices. Collected from Alibaba's production clusters within a one-hour time frame[1], these records span a network of ten thousand bare-metal nodes [9]. This dataset serves as raw data for our study to find the relations of microservices categorization within individual services and extracting similarities between service's call graphs.

Our theoretical framework draws on social network analysis (SNA) to examine microservices architectures, utilizing methods to analyze relationships among interacting units and understand the complex web of service interactions. By employing community detection algorithms, and graph similarity techniques, we identify closely interconnected groups of microservices and categorize the relationships between different services. This approach allows us to uncover patterns of collaboration and dependency within microservices networks, offering insights into system optimization and performance enhancement in industrial contexts.

The main contribution of this paper is the application of social network analysis techniques to analyze the complex interactions within microservices architectures at an industrial scale. We employ community detection algorithms and graph similarity measures to reveal patterns of microservice interconnections. This method enhances our understanding of microservices dynamics, offering a framework for improving system design, performance, and fault tolerance. Our research provides actionable insights for system managers and developers, aiming to improve software architectures' resilience and efficiency in real-world applications.

## 2 RELATED WORKS AND BACKGROUND

Exploring the architecture of services and microservices, along with discovering patterns and new observations, can help in different aspects of system application. Studies [3, 10] have explored the structural complexities of microservices, employing graph-based techniques to map the complex web of service interactions. Gaidels et al. [6] emphasizes the significance of employing graph-based

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651842

[1] https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2022

techniques in system analysis. Similarly, S. Luo's examination of the same dataset [8] indicates that identifying similarities within service call graphs can offer valuable insights into the characterization of microservice dependencies and their runtime performance.

These investigations underscore the potential of network analysis in identifying performance bottlenecks and optimizing service orchestration. Furthermore, study [5] highlights the significance of community detection in uncovering latent patterns within microservices networks, suggesting that such methodologies can facilitate a deeper understanding of service dependencies and interactions.

Community detection, a fundamental challenge in network analysis, involves categorizing nodes into distinct groups based on connections, typically focusing on structural aspects [4]. The Louvain method is an algorithm for identifying aggregate connected groups of nodes within a graph. Its primary strength lies in its ability to reveal the underlying modular structure within a network, emphasizing that nodes within the same module share more connections with each other than with nodes outside the module.

Building upon these foundations, our research introduces a novel approach by integrating community detection with clustering algorithms to analyze microservices at an industrial scale. Our study leverages the Louvain method for community detection and the K-means algorithm for clustering, aiming to provide actionable insights into microservices categorization and the optimization of service call graphs.

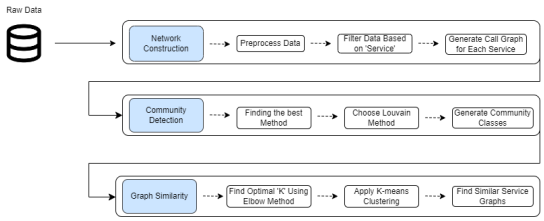



**Figure 1: Our system design diagram**

## 3 METHODOLOGY

In this section, we explain our methodology, outlined through three sequential steps as depicted in Figure 1. The main idea is to analyze the relationships among microservices within a singular service architecture by applying the community detection methods as well as analysing the similarities by clustering service graphs, which is the graphs of the microservices inside a unique service.

We start this process by making an accurate dataset for our methodology. This initial phase involves the gathering and preparation of data, ensuring its preparation and conducting subsequent analysis. In the second step, we apply a community detection algorithm to the prepared data. This critical phase plays a key role in categorizing each microservice within a given service into its designated community class, exposing the interconnections. The algorithm identifies patterns and relationships, providing a structured framework for understanding the complex web of associations among microservices.

In the last step, we apply a method that involves an examination of similarities within various service graphs. To find these similarities, we use a clustering method on different service graphs to group them. By analyzing these clusters, we gain insights into the dynamics of different services. Having the results of the second and third steps together can give us an internal observation of the whole system.

### 3.1 Network Construction

In the present investigation, the Alibaba production clusters data was employed; however, it is necessary to apply preprocessing to construct the network we needed for the rest of our methodology. Our investigation centered on a 1-hour snapshot of this dataset, consisting of over 260 million records and involving more than 28,000 microservices. As depicted in Figure1, this stage includes three sub-steps: preprocessing, filtering, and generating call graphs. Within this phase, the raw data serves as input, and the resulting output consists of graph datasets suitable for the application of community detection and graph similarity methods.

In the preprocessing step, we reduced the unnecessary attributes in the raw data and kept just three columns, focusing on the specific data points crucial for our analysis. We will do this reduction due to the efficiency we will gain from our method with this modified dataset as it would make the methods functioning faster and easier. The resulting dataset included 'um' and 'dm' representing the 'upper microservice' and 'down-stream microservice' in the call request, serving as the foundation for constructing call graph nodes and edges. The 'service' attribute was also retained, providing valuable information for generating the call graph of each service. Rows containing attributes with unacceptable values were omitted.

Following the preprocessing phase, we organize datasets for each service independently by implementing data filtering based on the 'service' attribute. Considering the vast size of our dataset, we opted to focus on a representative sample of all services. As services with a higher number of call requests have a more complex and interesting call graph, we applied a filter to include only service call graphs with a minimum of 50 call requests and randomly selected 300 of them for further analysis.

Following the filtering step, we obtained 300 dataframes, each representing a call graph for a unique service. Within these service datasets, individual rows contain the names of two microservices, with one calling the other on a specific service. Each microservice is treated as a node in the call graph, and if two nodes are present in a single row, a connection is established, resulting in an edge between the corresponding microservice nodes in the call graph. These output datasets facilitate an analysis of the interconnections and dependencies within each service.

### 3.2 Community Detection

Utilizing the provided call graph dataset, the subsequent step involves creating community classes and assigning each microservice of a service to the appropriate community class. Community detection methods are mainly designed to recognize groups or communities within a network, focusing on the patterns of connections between nodes. In our case, the goal is to demonstrate the relationships between nodes based on their services, we applied the

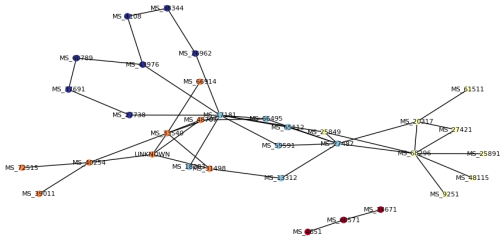



**Figure 2: Community classes on a service call graph**

community detection method to each service dataset. This process facilitated the identification and categorization of microservices into distinct community classes.

As shown in Table 1, different community detection methods were available for application. To identify the most suitable approach for our dataset, we conducted an evaluation using a sample exceeding 13 million records from the main dataset. The four methods employed for evaluation included the Greedy Modularity method, the Louvain method, the Info-map method, and the Label-propagation method. The assessment was based on key metrics, namely the Coverage, Modularity score, and Silhouette score. Coverage measures the comprehensiveness of community detection methods. In this study, all of the chosen methods have a coverage metric of 1. Modularity score measures the quality of identified community structure. The Silhouette Score, on the other hand, evaluates the cohesion and separation of clusters. These metrics collectively enable an evaluation of the efficacy and quality of community detection algorithms in network analysis. The detailed results of this evaluation are presented in Table 1.

| Methods | Silhouette Score | Modularity Score |
|---|---|---|
| Greedy Modularity | 0.65 | 0.60 |
| Louvain | 0.71 | 0.67 |
| Info-map | 0.58 | 0.61 |
| Label-propagation | 0.63 | 0.58 |

**Table 1: Evaluation scores for different methods.**

As evident in Table 1, the Louvain method has a better performance compared to the other methods under consideration. The Louvain method serves as a robust technique for detecting communities or clusters in complex networks. Its primary goal is revealing the underlying modular structure within a network, emphasizing that nodes within the same module share more connections with each other than with nodes outside the module. As microservices architectures often involve numerous interconnected components, the Louvain method's scalability becomes crucial in handling large-scale networks. Recognized as an enhanced version of the Greedy Modularity algorithm, the Louvain method utilizes a random seed as the foundation for its calculations. To ensure consistent results across different code iterations, we opted to fix the random seed value. To determine the optimal random seed, we systematically varied the seed value from 0 to 1000, selecting the value that yielded the highest Modularity score based on our evaluation criteria. This approach ensured stability and reliability in our community detection results.

To achieve the final results, we applied the Louvain algorithm to the refined dataset obtained in the previous step. The output consists of community classes for each service graph, and Figure 2 displays graphical representations for one selected service call graph. In this plot, nodes represent microservices, and the color of each node signifies its assigned community class. In this picture, nodes in close proximity often share the same community class, reflecting the method's ability to capture patterns in network connections. The color-coded representation facilitates the clear identification of distinct community classes within each service graph. Furthermore, the number of callings between nodes emerges as an essential factor influencing node assignments, providing insights into complex network dynamics and inter-service dependencies captured by the Louvain method.

## 3.3 Graph Similarity

Building on the previous steps about how microservices relate to each other inside a service, exploring similar service graphs can provide another viewpoint into the entire system. We applied the K-means method to a bunch of service datasets to pinpoint service graphs with similar characteristics. The K-means algorithm uses distinctive features from each graph and then organizes them into 'k' clusters based on these features. This approach helps the categorization of service graphs, providing a deep understanding of the system's diverse patterns and structures. K-Means can handle large datasets and is computationally efficient, making it scalable for systems with a significant number of services. Also, microservices communication patterns may not follow a strict shape, and K-Means can be robust in identifying clusters with irregular shapes. So, in general, this method brings us well to our intended goals for this study.

Within this study, our approach involves clustering graphs based on their structure. Specifically, we extract nodes and edges from each graph, considering them as features important for the clustering process. An additional challenge is in determining the most suitable value for 'k' in this dataset. To address this, we employed the Elbow method [1], which involves testing various values of 'k' within the K-means method and plotting the distribution. Analyzing the plot obtained from the Elbow method, we identified a distinct bend or "elbow," and based on this observation, we determined that setting 'k' to 5 yielded optimal results for our dataset.

Following the execution of K-means on the dataset, we obtained 5 clusters, each including a certain number of graphs. Due to space constraints, we present four out of the five clusters identified in our analysis in Figure 3. To assess the effectiveness of this method, we utilized the Silhouette score. The Silhouette score obtained for our method was 0.6141, indicating a desirable performance for this type of clustering.

As the conclusion of our efforts, the final results yield a series of graphs, providing enhanced insights into the diverse services within the system and their respective structures. In Figure 3, services are organized into distinct clusters based on their overall structure. As a sample, graphs '3a' and '3b' form one cluster, '3c' and '3d' another,

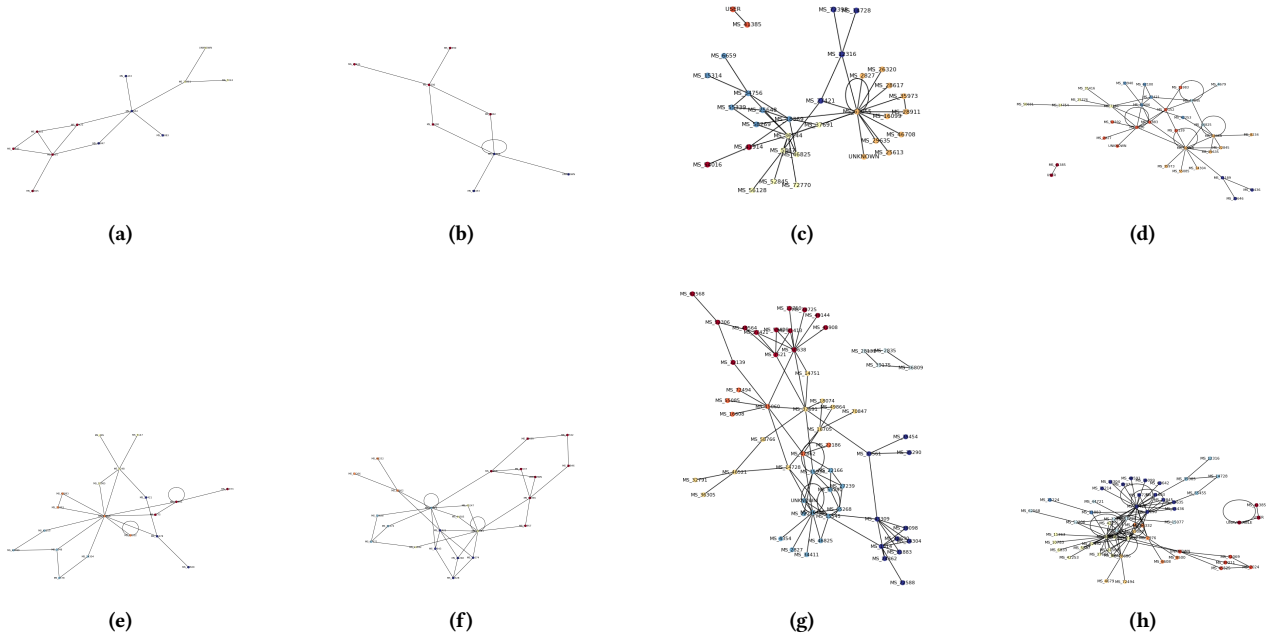


**Figure 3: Similarity Clusters on Service Graphs**

'3e' and '3f' a third, and '3g' and '3h' constitute a fourth cluster. This analysis contributes to a deeper understanding of the actual runtime configurations present in the system's services. The source codes of the implementations are available from our anonymized GitHub repository [2].

## 4 DISCUSSION

Upon applying community detection to the randomly selected 300 services, we observed instances where certain services contained only two microservices. In these cases, both microservices would be assigned to the same community class. However, as the service graph complexity increased, there would be more microservices and community classes, and finding a strong connection among microservices within the same class would be worth more. As you can observe in Figure 3, more complex graphs, will cause a heightened frequency of calls and more direct communication between microservices within a class. By identifying these relationships, we can pinpoint areas of latency or bottlenecks in the system. In case of issues, detecting the faulty microservice and its collaborators becomes more feasible. Additionally, this knowledge enables effective isolation of issues, preventing them from affecting other parts of the system. A clear understanding of how different microservices interact during development and debugging is essential. Developers require insights into data flow, dependencies, and communication protocols between microservices to write effective code and troubleshoot efficiently.

In this study, we pursued the identification of similar service graphs based on their microservice connections. This approach provides several advantages for system management and decision-making. One such benefit is evident in resource allocation and scaling, where similar services often exhibit comparable patterns in resource usage and demand. Efficient resource allocation becomes possible by recognizing these similarities. Performance benchmarking is another advantage, allowing for comparisons of microservice relations and performance metrics among similar services. This benchmarking process helps us continually improve and make services work better in the whole system. Importantly, this perspective enhances decision-making by highlighting the similarities among services, playing a critical role in strategic considerations for the system's development.

[2]https://github.com/ghazalkhb/ICPE2024_DataChallenge

## 5 CONCLUSIONS AND FUTURE WORK

We conducted an analysis of the relationships and communications among microservices within a service, as well as the similarities between different services. This achievement was realized through the application of community detection on microservices within a service, followed by clustering the resulting service graphs. The outcomes of this investigation provide numerous advantages for system performance and management.

While the current study focused on existing attributes, the inclusion of additional attributes for each node or service could further enhance our insights. Future works could involve incorporating response times for each connection, offering a more delicate understanding of the system's dynamics. Furthermore, scaling up the study to a larger dataset would provide a broader observation of system functionality. Another avenue for exploration is the examination of additional methods for community detection and clustering algorithms or even the combination of multiple methods. Integrating machine learning techniques to predict future dynamic network behaviors and community formations represents another promising direction.

# REFERENCES


[1] Purnima Bholowalia and Arvind Kumar. 2014. EBK-means: A clustering technique based on elbow method and k-means in WSN. *International Journal of Computer Applications* 105, 9 (2014).

[2] Grzegorz Blinowski, Anna Ojdowska, and Adam Przybyłek. 2022. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. *IEEE Access* 10 (2022), 20357–20374. https://doi.org/10.1109/ACCESS.2022.3152803

[3] Álvaro Brandón, Marc Solé, Alberto Huélamo, David Solans, María S Pérez, and Victor Muntés-Mulero. 2020. Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software* 159 (2020), 110432.

[4] Petr Chunaev. 2020. Community detection in node-attributed social networks: a survey. *Computer Science Review* 37 (2020), 100286.

[5] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Xiaomin Wu, Meng Zhang, Qingjun Chen, Xin Gao, Xuedong Gao, et al. 2023. TraceDiag: Adaptive, Interpretable, and Efficient Root Cause Analysis on Large-Scale Microservice Systems. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1762–1773.

[6] Edgars Gaidels and Marite Kirikova. 2020. Service dependency graph analysis in microservice architecture. In *Perspectives in Business Informatics Research: 19th International Conference on Business Informatics Research, BIR 2020, Vienna, Austria, September 21–23, 2020, Proceedings 19*. Springer, 128–139.

[7] Shanshan Li, He Zhang, Zijia Jia, Chenxing Zhong, Cheng Zhang, Zhihao Shan, Jinfeng Shen, and Muhammad Ali Babar. 2021. Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. *Information and Software Technology* 131 (2021), 106449. https://doi.org/10.1016/j.infsof.2020.106449

[8] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*. 412–426.

[9] Shutian Luo, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Guodong Yang, and Chengzhong Xu. 2022. The power of prediction: microservice auto scaling via workload learning. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 355–369. https://doi.org/10.1145/3542929.3563477

[10] Vinay Raj and Ravichandra Sadam. 2021. Evaluation of SOA-based web services and microservices architecture using complexity metrics. *SN Computer Science* 2 (2021), 1–10.

[11] Victor Velepucha and Pamela Flores. 2023. A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE Access* (2023).

# Unveiling Temporal Performance Deviation: Leveraging Clustering in Microservices Performance Analysis

André Bauer
University of Chicago
Chicago, United States

Timo Dittus
University of Würzburg
Würzburg, Germany

Martin Straesser
University of Würzburg
Würzburg, Germany

Alok Kamatar
University of Chicago
Chicago, United States

Matt Baughman
University of Chicago
Chicago, United States

Lukas Beierlieb
University of Würzburg
Würzburg, Germany

Marius Hadry
University of Würzburg
Würzburg, Germany

Daniel Grillmeyer
University of Würzburg
Würzburg, Germany

Yannik Lubas
University of Würzburg
Würzburg, Germany

Samuel Kounev
University of Würzburg
Würzburg, Germany

Ian Foster
Argonne National Laboratory
Lemont, United States

Kyle Chard
University of Chicago
Chicago, United States

## ABSTRACT

As the market for cloud computing continues to grow, an increasing number of users are deploying applications as microservices. The shift introduces unique challenges in identifying and addressing performance issues, particularly within large and complex infrastructures. To address this challenge, we propose a methodology that unveils temporal performance deviations in microservices by clustering containers based on their performance characteristics at different time intervals. Showcasing our methodology on the Alibaba dataset, we found both stable and dynamic performance patterns, providing a valuable tool for enhancing overall performance and reliability in modern application landscapes.

## CCS CONCEPTS

• **General and reference** → **Performance**; **Measurement**.

## KEYWORDS

Temporal Analysis, Clustering, Performance Analysis, Performance Deviation, Microservices, Data Challenge

**ACM Reference Format:**
André Bauer, Timo Dittus, Martin Straesser, Alok Kamatar, Matt Baughman, Lukas Beierlieb, Marius Hadry, Daniel Grillmeyer, Yannik Lubas, Samuel Kounev, Ian Foster, and Kyle Chard. 2024. Unveiling Temporal Performance Deviation: Leveraging Clustering in Microservices Performance Analysis. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3651843

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651843

## 1 INTRODUCTION

As enterprises increasingly transform their applications into microservices and deploy them in cloud environments [3], the orchestration of hundreds of these microservices becomes crucial in shaping the performance and reliability of applications. The complexity of this endeavor is illustrated, for example, by Alibaba's production infrastructure, which contains more than 28,000 microservices and 400,000 containers [6]. Consequently, managing such a large number of microservices and containers is a major challenge, particularly when potential performance degradations or failures should be detected. In this context, fine-grained performance analysis and monitoring becomes a daunting task.

To facilitate the performance monitoring of microservices, we propose a methodology to reveal the temporal performance deviations of microservices. The overarching concept involves clustering microservice containers according to their performance attributes at different time periods. By analyzing these temporal clusters, we can pinpoint deviations in performance behavior. In other words, over time, microservice containers[1] may shift between clusters, enabling the identification of containers that exhibit notable performance deviations. For example, if a certain percentage of microservice containers exhibit such variations, this could be the trigger to investigate why that particular microservice is behaving differently, ultimately helping to ensure the overall performance and reliability of the system. With the temporal dimension, these insights could be mapped to possible root causes (e.g., abnormal user behavior at runtime or an application update that degraded the performance).

We utilized the Alibaba dataset [6], which contains runtime information of microservices and the underlying hardware, to gain valuable insights and showcase our methodology. Our investigation[2] showed that the performance of some microservices remains stable the whole time, while others change their performance significantly.

[1] A container serves as an instance of a microservice.
[2] The analysis is available at CodeOcean https://doi.org/10.24433/CO.6129510.v1.

The remainder of this paper is structured as follows: In Section 2, we provide information on the analyzed dataset, clustering, and related work. In Section 3, we present our methodology. In Section 4, we investigate the dataset and the performance of our methods. In Section 5, we conclude the paper.

# 2 BACKGROUND

## 2.1 Dataset Description

Luo et al. [6] released the Alibaba dataset used in this paper. The dataset includes detailed runtime metrics derived from microservices operating in Alibaba's production infrastructure. Collected over 13 days in 2022 from the Alibaba Cloud, these traces are categorized into four parts: *Node*, which captures bare-metal node runtime information; *MSResource*, responsible for documenting CPU and memory utilization of containers from different microservices; *MSRTMCR*, providing details on microservice call rate and response time; and *MSCallGraph*, encompassing call graphs among microservices across multiple clusters. The dataset covers over 400,000 bare-metal nodes, 28,000 microservices, and 470,000 containers. Our approach specifically concentrates on the information gathered by *MSResource*.

## 2.2 K-Means & Silhouette Coefficient

*K-means* [5, 7] is a clustering algorithm used in machine learning and data analysis. It aims to partition a dataset into $k$ distinct, non-overlapping subgroups (clusters) based on the similarity of data points. The algorithm iteratively assigns each data point to the cluster with the nearest centroid (i.e., geometric center) and updates the centroids accordingly. One challenge in k-means is determining the optimal number of clusters $k$ for a given dataset.

The *silhouette coefficient* is a metric that quantifies how well-separated the clusters are. Essentially, it measures the similarity of an object to its own cluster (cohesion) as opposed to other clusters (separation). It ranges from -1 to 1, with higher values indicating better-defined clusters. A silhouette coefficient close to 1 suggests that data points within a cluster are more similar to each other than to those in other clusters. This metric is valuable for assessing the quality of clustering results and guiding the choice of an optimal number of clusters in the k-means algorithm.

## 2.3 Other Applied Methods

*Principal Component Analysis* (PCA) [8] is a dimensionality reduction technique. The primary goal of PCA is to transform high-dimensional data into a lower-dimensional representation, capturing the most significant variations in the original dataset. It identifies the principal components, orthogonal axes along which the data exhibits maximum variance.

A *Decision Tree* [2] is an intuitive and powerful tool in machine learning and data analysis commonly employed for classification and regression tasks. These hierarchical structures recursively split the dataset into subsets based on the most informative features, resulting in a tree-like model of decisions. The tree evaluates a specific feature at each internal node, choosing the split that optimally separates the data. The process continues until the creation of leaf nodes, each representing a distinct class or a numerical value.

## 2.4 Related Work

To the best of our knowledge, there are few publicly accessible trace datasets from production systems made available and investigated. For instance, Azure explored and released a serverless dataset containing function invocation and execution times in 2019 [9]. In a more recent study, Azure presented a similar dataset [10]. A further example is the dataset utilized in this paper, released by Alibaba [6], containing runtime metrics of microservices. In another study [1], a scientific serverless dataset was investigated and released, encompassing function invocations, characteristics of functions, and function execution times. Meta also released and investigated a dataset [4], containing topology and call information of their microservices. In contrast to these works, we do not release a dataset but utilize the aforementioned Alibaba dataset to explore the temporal behavior deviation of the microservices.

# 3 APPROACH

## 3.1 High-Level Idea

To examine the temporal performance deviation of microservices, we partition the Alibaba dataset into discrete time intervals. Within each dataset segment, we capture the performance attributes of individual microservice containers, enabling comparison across distinct time periods. Please note that the dataset comprises diverse microservices, each implemented with various deployed containers. To accomplish this, we employ clustering and compute an initial cluster based on the first temporal segment. We then assign each microservice container to a cluster based on its performance characteristics. Subsequently, for each segment, we reassign microservices containers to clusters. This methodology enables identification of microservices whose containers exhibit significant performance deviations, as reflected by switching clusters. For our comparative analysis (see Section 4), we have chosen a time interval of one hour, thus dividing the Alibaba dataset into hourly segments. However, it is important to note that this interval can be adjusted as needed to offer a more refined or broader resolution.

## 3.2 Performance Characteristics Extraction

To cluster the different microservices containers, information from each time interval has to be extracted. Within a given dataset segment, we identify all containers operating during this specific time frame. Subsequently, for each container, we collect CPU and memory utilization data, as well as the normalized runtime (i.e., the actual runtime within this interval divided by the length of the interval). The CPU and memory utilization data are then aggregated into statistical summaries, including the mean, standard deviation, and quartiles (minimum, 25th percentile, median, 75th percentile, and maximum). Finally, these aggregated performance metrics, along with the runtime information, are utilized for the clustering.

## 3.3 Performance Clustering

To cluster the containers from microservices according to their performance characteristics, we employ the k-means algorithm and the elbow method to determine the optimal number $k$ of clusters. Specifically, we applied clustering for $k \in \{2, 3, \ldots, 10\} \cup \{25, 50, 100\}$ and

utilized the silhouette coefficient to determine the optimal value for $k$ which is 5.

Figure 1 illustrates the decision-making process of k-means. The depicted decision tree is truncated and is, therefore, only an approximation of the clustering. For instance, all microservice containers with a mean memory utilization of less than 0.29 were grouped into Cluster 3. Another illustration is Cluster 1, encompassing microservice containers with a mean memory utilization greater than or equal to 0.29 but less than 0.54 and a mean CPU utilization less than 0.31. The rules for the remaining clusters are more intricate. Please note that the presented values are derived from the hourly aggregation. Specifically, the initial split in the decision tree checks whether a microservice container encountered an average memory consumption higher than 0.54 during the hour.

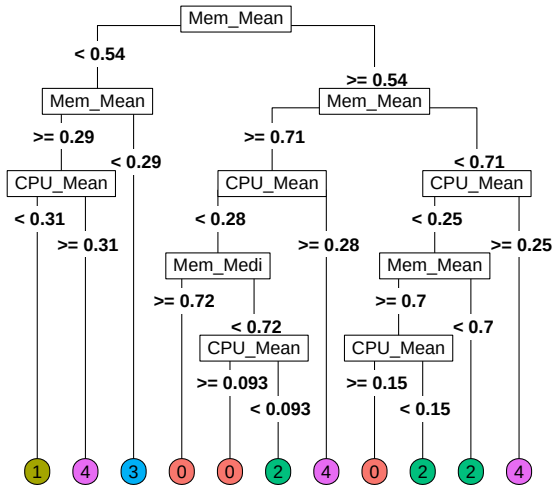



**Figure 1: Explanation of the clustering.**

## 3.4 Possible Extensions and Discussion

Our methodology is designed to work on the CPU and memory utilization to keep the approach generic—that is, supporting any application. To tailor it to a specific use case, one could incorporate application-level information such as the read/write rate to/from a database, the length of the queue, the number of active threads, etc. Please remember that values must be normalized between 0 and 1 to ensure that the value range of each feature for the clustering is equal. Of course, the aggregation time can be set to any arbitrary time to offer a more refined or broader resolution and does not have to be exactly one hour. Additionally, our initial findings indicate that k-means outperformed hierarchical and density-based clustering methods on the Alibaba dataset. However, it is important to acknowledge that the effectiveness of these clustering algorithms may vary depending on the dataset or input being utilized.

# 4 EXPLORATION OF THE DATASET

## 4.1 Data Selection

Given the extensive size of the dataset, we carefully chose different points in time from the first, second, and final days to thoroughly examine microservices' performance deviation. In detail, our analysis encompassed the first hour of the first day (denoted as *d0h0* to match Alibaba's syntax). Subsequently, we investigated the second hour to capture immediate changes (referred to as *d0h1*). For a greater temporal span, we delved into the 13th hour (referred to as *d0h12*) to assess performance half a day later. Additionally, we examined the first hour of the second day to compare performance a full day later (named as *d1h0*). Finally, we extended the investigation to the last hour of the last available day (labeled as *d?h23*).

Although the dataset covers 13 days, the download script provided by Alibaba aborts after the dataset part with the number 1008. Considering the file naming convention, the file corresponds to the last hour of day 3. However, upon inspecting the timestamps of the last downloaded file, they fall within the final hour of day 21. In essence, we lack clarity on the date to which this file pertains. Consequently, we assign the label *d?h23* to denote the uncertainty regarding the date.

**Table 1: Overview of the number of microservices and their containers at selected points in time.**

| | *d0h0* | *d0h1* | *d0h12* | *d1h0* | *d?h23* |
|---|---|---|---|---|---|
| #unique MS | 28,164 | 28,167 | 28,173 | 28,013 | 27,707 |
| #MS containers | 467,822 | 467,004 | 472,320 | 480,646 | 447,353 |

An overview of the number of individual microservices and the total number of containers at the selected points in time is listed in Table 1. The quantity of unique microservices and their containers fluctuates between different time points. For example, comparing *d0h0* and *d?h23*, the dataset reveals a decrease of 457 microservices and 20,469 containers. Moreover, the number of shared unique microservices (see Table 2) stands at 28,161, 28,143, 27,834, and 27,650 between *d0h0* and *d0h1*, *d0h0* and *d0h12*, and so forth, respectively.

## 4.2 Clustering the Dataset

To assess the effectiveness of the microservice clustering, we analyzed all 28,164 microservices, examining the distribution of their containers across clusters. Notably, 22,873 microservices demonstrated a homogeneous distribution, with all containers residing within a single cluster, while 5,291 exhibited a distribution across multiple clusters.

In addition to quantitative analysis, we employed visualizations to evaluate the clustering quality of microservice containers. Employing PCA to reduce the dimensionality, we plotted the clustered containers on two principal axes, as depicted in Figure 2. This approach revealed five distinct areas, each representing a cluster. While most microservice containers aligned with their respective clusters, a few outliers were observed, such as the three blue dots in the golden area. Overall, we are confident in the clustering's ability to distinguish and classify various microservices' performance behavior accurately.

## 4.3 Temporal Performance Deviation

To investigate the temporal performance deviation of the microservices, we compare the shift of microservice containers between

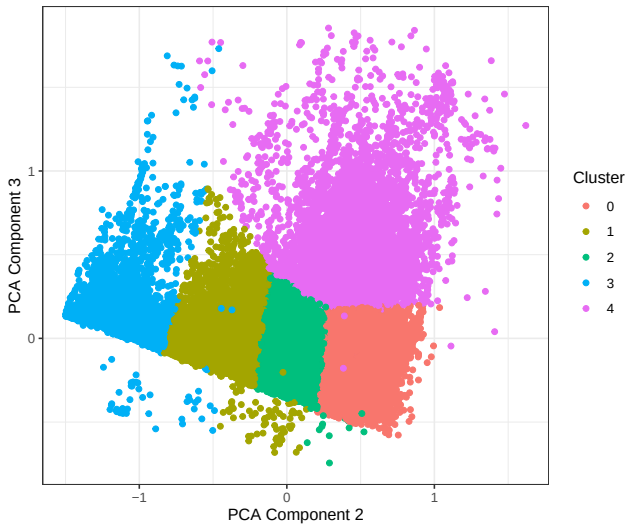



**Figure 2: Visualization of the clustering.**

clusters over the selected periods in time. To this end, we only consider containers that exist across all periods in time.

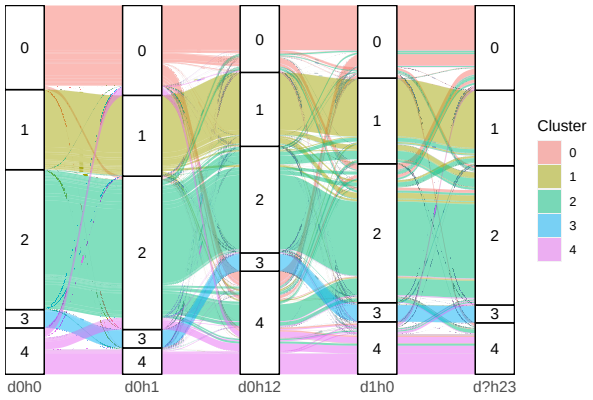



**Figure 3: Visualization of the temporal performance deviation through the shift of microservices between clusters over time. The color of each flow represents the initial cluster.**

We start with a visual assessment in Figure 3, where the color of the flows represents the cluster the microservice containers originate from—that is, from the initial cluster derived from *d0h0*. Over time, a noticeable evolution occurs in the number of containers within each cluster, as illustrated by the varying heights of boxes labeled with their respective cluster IDs. The most substantial change is observed in Cluster 4. While the container count remains relatively stable (50,806, 57,486, 56,374) for *d0h0*, *d1h0*, and *d?h23*, it undergoes a drastic reduction from *d0h0* to *d0h1* (28,908) and subsequently quadruples from *d0h1* to *d0h12* (112,913). In contrast, Cluster 3 exhibits the slightest deviation, with nearly all containers persisting within this cluster throughout the entire duration, visually depicted by the blue flow, which almost has no forks. In numerical terms, out of the initial 20,056 containers clustered into Cluster 3, 18,101 containers remain within this cluster during the selected time points. That is, these containers do not experience a significant change in their temporal performance behavior.

In addition to the visual analysis, we employ a descriptive investigation listed in Table 2. Initially, we explore the diminishing count of unique microservices and containers shared with the initial time point (i.e., *d0h0*), indicating potential changes in the functionality of Alibaba's production infrastructure over time. At the same time, the performance changes, evident in the decreasing number of microservices maintaining their cluster distribution, that is, containers remaining in the same clusters across selected time points, with percentages of 91%, 80%, 72%, and 71% from *d0h1* to *d?h23*. The change is further illustrated by the percentage of microservices where all containers change their clusters, which stands at 1%, 4%, 8%, and 9%. When comparing pairwise performance deviations, the most prevalent pairs, ranked by decreasing similarity, are (*d0h0*, *d0h1*), (*d0h0*, *d1h0*), (*d0h0*, *d?h23*), and (*d1h0*, *d?h23*).

**Table 2: Overview of the temporal performance deviation. The clusters were derived based on *d0h0*, that is, Rows 3–7 are computed based on this initial clustering.**

| | *d0h1* | *d0h12* | *d1h0* | *d?h23* |
|---|---|---|---|---|
| #shared MS with *d0h0* | 28,160 | 28,132 | 27,537 | 27,132 |
| #shared MS containers with *d0h0* | 466,745 | 465,422 | 428,036 | 404,762 |
| #MS kept cluster distribution | 25,511 | 22,592 | 19,936 | 19,139 |
| #MS containers stayed in same cluster | 432,408 | 365,330 | 345,957 | 326,242 |
| #MS changed cluster distribution | 2,649 | 5,540 | 7,601 | 7,993 |
| #MS containers changed cluster | 34,337 | 100,092 | 82,079 | 78,520 |
| #MS where all containers changed cluster | 226 | 1,056 | 2,318 | 2,552 |

## 5 CONCLUSION

Our proposed methodology for monitoring the temporal performance deviations of microservices offers a valuable solution to the challenging task of managing large-scale microservice deployments. This enables lightweight identification of containers with notable performance variations, which can subsequently undergo in-depth investigation through more intricate analyses to guarantee the overall performance and reliability of the system. We revealed stable and dynamically changing performance patterns while applying our methodology to the Alibaba dataset.

## ACKNOWLEDGEMENT

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 510552229. This work was partially supported by NSF 2004894 and Argonne National Laboratory under U.S. Department of Energy under Contract DE-AC02-06CH11357.

# REFERENCES


[1] André Bauer, Haochen Pan, Ryan Chard, Yadu Babuji, Josh Bryan, Devesh Tiwari, Ian Foster, and Kyle Chard. 2024. The Globus Compute Dataset: An Open Function-as-a-Service Dataset From the Edge to the Cloud. *Future Generation Computer Systems* 153 (4 2024), 558–574. https://doi.org/10.1016/j.future.2023.12.007

[2] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. 1984. *Classification and Regression Trees*. Chapman and Hall/CRC.

[3] European Statistical Office (Eurostat). 2021. Cloud computing - statistics on the use by enterprises. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises Retrieved October 13, 2022.

[4] Darby Huye, Yuri Shkuro, and Raja R. Sambasivan. 2023. Lifting the veil on Meta's microservice architecture: Analyses of topology and request workflows. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 419–432. https://www.usenix.org/conference/atc23/presentation/huye

[5] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[6] Shutian Luo, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Guodong Yang, and Chengzhong Xu. 2022. The Power of Prediction: Microservice Auto Scaling via Workload Learning. In *Proceedings of the ACM Symposium on Cloud Computing*.

[7] James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[8] Karl Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2, 11 (1901), 559–572.

[9] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX Annual Technical Conference*. USENIX Association, 205–218.

[10] Yanqi Zhang, Íñigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. 2021. Faster and Cheaper Serverless Computing on Harvested Resources. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 724–739. https://doi.org/10.1145/3477132.3483580

# Grammar-Based Anomaly Detection of Microservice Systems Execution Traces

Andrea D'Angelo
andrea.dangelo6@graduate.univaq.it
DISIM Department
University of L'Aquila
L'Aquila, Italy

Giordano d'Aloisio
giordano.daloisio@graduate.univaq.it
DISIM Department
University of L'Aquila
L'Aquila, Italy

## ABSTRACT

Microservice architectures are a widely adopted architectural pattern for large-scale applications. Given the large adoption of these systems, several works have been proposed to detect performance anomalies starting from analysing the execution traces. However, most of the proposed approaches rely on machine learning (ML) algorithms to detect anomalies. While ML methods may be *effective* in detecting anomalies, the training and deployment of these systems as been shown to be less *efficient* in terms of time, computational resources, and energy required.

In this paper, we propose a novel approach based on Context-free grammar for anomaly detection of microservice systems execution traces. We employ the SAX encoding to transform execution traces into strings. Then, we select strings encoding anomalies, and for each possible anomaly, we build a Context-free grammar using the Sequitur grammar induction algorithm. We test our approach on two real-world datasets and compare it with a Logistic Regression classifier. We show how our approach is more effective in terms of training time of ∼15 seconds with a minimum loss in effectiveness of ∼5% compared to the Logistic Regression baseline.

## CCS CONCEPTS

• **Software and its engineering → Software performance**; *Formal language definitions*; • **Theory of computation → Grammars and context-free languages**.

## KEYWORDS

Anomaly Detection,Execution Traces,Context-free Grammar,Micro Service System

**ACM Reference Format:**
Andrea D'Angelo and Giordano d'Aloisio. 2024. Grammar-Based Anomaly Detection of Microservice Systems Execution Traces. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3651844

## 1 INTRODUCTION

Microservice architecture is nowadays one of the most adopted architectural patterns to develop large-scale systems (like Netflix, Amazon, Facebook, and others) [1]. In general, a microservice system can be represented as a network of individually deployed systems, each one devoted to a single specific task (i.e., a *microservice*). By interacting with each other, the different microservices allow the completion of more complex tasks for the end user. Given the wide adoption of this architectural style, several studies have been conducted to tackle performance anomalies of these kinds of systems. In particular, anomaly detection of microservice systems is a widely addressed topic in the literature [26]. However, most of the methods proposed employ machine learning (ML) algorithms to address this task. While the adoption of these approaches can be *effective* in terms of anomaly detection, the training and deploying of ML methods is generally not *efficient* in terms of required training time, computational resources, and energy consumption [11, 18, 21].

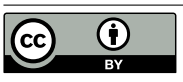
In this work, we move towards a more energy-efficient anomaly detection in microservice systems execution traces by presenting an innovative approach based on formal Context-Free grammar. We employ SAX encoding [25] to transform execution traces into strings, and then infer a grammar from the set of strings that encode anomalies. The constructed grammar functions as an anomaly detector, enabling the encoding and membership check of any new measurement. We present a first implementation using Sequitur [22] for grammar induction. We compare the training time and effectiveness scores of our approach against Logistic Regression. Results show that the grammar-based approach achieves comparable effectiveness while requiring significantly less time for training.

The remainder of the paper is structured as follows: in Section 2 we discuss some related works; Section 3 presents in detail the proposed approach; Section 4 shows the experimental evaluation we conducted to assess the *efficiency* (in terms of required time) and *effectiveness* of our approach; finally Section 5 presents some future works and concludes the paper.

## 2 RELATED WORKS

The problem of performance analysis, and in particular, anomaly detection in the performances of microservice systems, has been widely studied by the literature [26].

Among all the proposed approaches, we observe how most of them employ ML techniques. For instance, Bensal *et al.* proposed DeCaf, a system for automated diagnosis and triaging of KPI issues using service logs [4]. The proposed approach uses machine learning along with pattern mining to help service owners automatically root cause and triage performance issues. Similarly, Du *et al.* presented a system for anomaly detection in the performances of container-based microservice systems [10]. The proposed approach



This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651844

consists of a monitoring module that collects the performance data of containers, a data processing module based on machine learning models and a fault injection module integrated for training these models. An approach employing several ML algorithms is the one proposed by Jin *et al.* [14]. The authors perform two different anomaly detection analyses in their work: invocation chain anomaly analysis based on robust principal component analysis and a single indicator anomaly detection algorithm. The single indicator anomaly detection algorithm comprises an Isolation Forest algorithm, a One-Class Support Vector Machine algorithm, a Local Outlier Factor algorithm, and the $3\sigma$ principle. Finally, Wu *et al.* employed a Deep Learning model for performance diagnosis in cloud-based microservice systems [29]. All the described approaches employ ML techniques, which may be effective but also require high computational resources and time for their training [21]. Moreover, ML models are often black-box or challenging to interpret [8].

A different approach for anomaly detection in microservice systems is the one proposed by Traini *et al.* [28]. In their work, the authors proposed a search-based approach for diagnosing performance issues in service-based systems. In our work, we move towards the same direction of not employing ML-based techniques to identify anomalies in the performances of microservice systems. In particular, we first transform each execution trace (i.e., a sequence of response times of remote calls to different microservices) into a string using the Symbolic Aggregate Approximation (SAX) encoding [25], which is a well-established technique for anomaly detection [6, 13]. Next, we employ a context-free-grammar-based approach to identify if the string representation of the execution trace contains an anomaly or not.

# 3 METHODOLOGY

In this section, we formally define our methodology for grammar-based anomaly detection. Figure 1 depicts the methodology in all its components. We first detail the methodology for Grammar Construction, then move onto the process of Membership Testing.

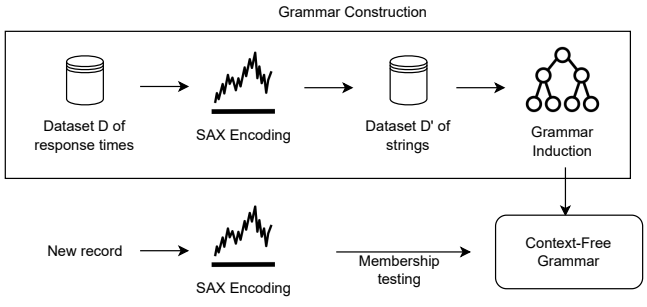



**Figure 1: The proposed methodology involves a one-time grammar construction process. Then, each new record undergoes membership checking against the grammar.**

## 3.1 Grammar Construction

We first focus on the one-time process of Grammar Construction. Starting from dataset D of response times, we define the SAX Encoding as the function $SAX : \mathbb{R}^n \rightarrow \Sigma^n$, where $\Sigma$ represents a predetermined set of characters. SAX takes a set of real numbers $i$ as input and maps them to a string of characters $s$ such that $|i| = length(s)$. By encoding all the response times in dataset D with SAX encoding, we obtain a dataset D' of labelled strings. From D' we select the execution traces (i.e., set of response times) showing anomalies. Then, for each anomaly category $A$, we apply a Grammar induction algorithm to the execution traces having that specific anomaly $A$, to obtain a Context-Free Grammar $G = (V, T, P, S)$, where V is the set of non-terminal symbols, T is the set of terminal symbols, P is the set of productions and S is the starting symbol. The grammar G must have the following properties:

(1) $T = \Sigma$
(2) $L(G) = \{SAX(i) \mid i \text{ is labeled as an anomaly}\}$

The first property ensures that the set of terminal symbols recognized by the grammar is the same as the set of characters obtained via SAX encoding. For this reason, the domain-specific choice of $\Sigma$ for SAX Encoding plays an important role in the resulting grammar. A $\Sigma$ encompassing numerous characters enhances the precision of anomaly detection, yet it may lead to an unwarranted increase in the size of the grammar. Conversely, a $\Sigma$ comprising only a few characters may not adequately discern subtle anomalies.

The second property fixes the language generated from G as the set of strings obtained via Sax Encoding and labelled as anomalies in the starting dataset D. Note that the second property does not imply that only those strings must be employed to build the grammar. Several grammar induction algorithms also consider negative examples as an aid to build the resulting grammar.

The grammar G was chosen to be Context-Free as it is able to represent intrinsic links inside of the string that is useful in our context. In instances where a microservice relies on several others, anomalies in their response times may be interconnected. It is crucial that the chosen grammar can accurately deduce that an elevated response time for one microservice could be contingent on another. Achieving this level of inference is not feasible with Regular Grammars or Regex, making the use of a Context-Free Grammar essential.

## 3.2 Membership Testing

Once the grammar G is built from the grammar induction algorithm, it effectively functions as a model for anomaly detection. We can now test new sets of real numbers for anomalies. When a new execution trace $j$ arrives, it must be processed with the same SAX function used for grammar construction. Then:

- if $SAX(j) \in L(G)$, j contains an anomaly.
- if $SAX(j) \notin L(G)$, there is no anomaly.

The process of understanding if a string is part of the language generated by a grammar is known as Membership checking. Several Python libraries provide convenient methods for this task (e.g., NLTK [5]).

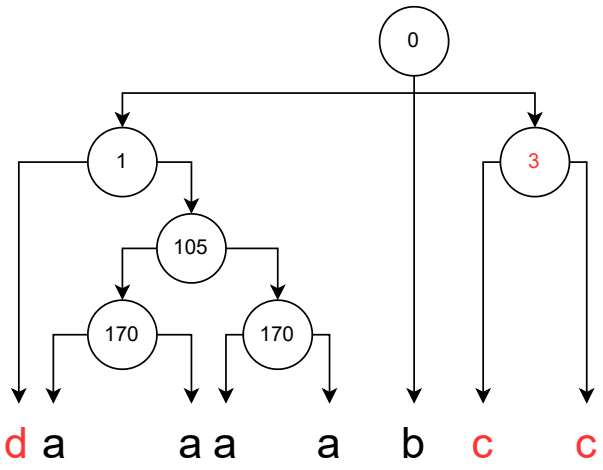



**Figure 2: Parse tree generated when checking the membership of a string that includes an anomaly in our grammar. The final two microservices exhibit values above the typical range, impacting the overall response time (initial character).**

```
0 -> 1 'b' 3
1 -> 'd' 105
105 -> 170 170
170 -> 'a' 'a'
170 -> 'a' 'a'
3 -> 'c' 'c'
```

**Listing 1: Grammar productions involved in the membership checking of string "daaaabcc".**

Whenever a string appears to be part of a grammar, the membership check also produces a parse tree of productions starting from the initial symbol of the grammar S to the given string. For instance, listing 1 depicts the necessary productions to derivate the string "daaaabcc" (which contains an anomaly) from the starting symbol of our grammar. The derivation can also be represented in a visually intuitive way by generating a parse tree. For instance, Figure 2 portrays a parse tree generated when the aforementioned string is processed by our grammar. The parse tree provides a visual representation that aids in understanding the precise locations of anomalies within the string. By examining the tree structure, we can easily pinpoint the specific steps and grammar productions leading to the anomalous elements.

## 4 EVALUATION

In this section, we describe the experimental evaluation conducted to assess the *efficiency* and *effectiveness* of our approach. In particular, we aim to answer the following two research questions (RQ):

**RQ1.** How much time does the proposed approach require to construct a grammar compared to the training time of standard ML methods?

**RQ2.** How effective is the proposed approach in detecting anomalies compared to standard ML methods?

In both experiments, we employ a Logistic Regression (LogReg) classifier [19] as a baseline. We have chosen this method among the possible classification approaches because it natively supports multi-class classification (i.e., classification problems where the number of possible values of the label is higher than two [2]) and because it usually achieves a good trade-off between prediction's effectiveness and training time compared to other classification models [17]. We adopted the implementation provided by the `scikit-learn` Python library [23].

Concerning our approach, we employ the Sequitur algorithm [22] for the grammar induction phase shown in figure 1. Sequitur is an algorithm that allows the generation of context-free grammars starting from a sequence of strings by replacing repeated phrases with a grammatical rule that generates the phrase. It repeats this process recursively until all the strings are examined.

As a use case, we employ the dataset provided by Traini *et al.* in [28][1]. The dataset comprises 560 CSV files containing pre-processed execution traces (i.e., series of response times of remote procedure calls to different microservices) with injected anomalies originating from two open-source microservices systems: Train-Ticket [16] and E-Shopper[2]. Each scenario features two possible anomalies, identified by the `anomaly` column.

In the following, we describe how we addressed **RQ1**. Next, we detail the answer to **RQ2**. The complete replication package of the experiment is available in Zenodo [9].

### 4.1 Addressing RQ1

The first research question focuses on the amount of time required by the proposed approach to generate context-free grammars compared to the amount of time needed for the LogReg classifier to train on a specific dataset. To answer this question, we computed the grammar construction and training times twenty times to avoid possible measurement biases. It is worth noticing how the grammar construction phase encompasses both the SAX encoding of the dataset and the grammar induction, as shown in figure 1. This experiment has been executed on a DELL XPS 13 2019 with an Intel i7 processor, 16 GB of RAM and Windows 11 operating system.

**Table 1: Comparison of means and standard deviation of Logistic Regression training and Grammar Construction times in seconds.**

| | E-Shopper | Train-Ticket |
|---|---|---|
| **Grammar** | 5.848 ± 0.506 | 7.724 ± 0.309 |
| **LogReg** | 18.833 ± 2.933 | 21.556 ± 2.863 |

Table 1 presents the mean and standard deviation of LogReg training time and grammar construction time for both E-Shopper and Train-Ticket datasets. As can be seen, our approach requires ~13 seconds less to construct a grammar compared to the training time of the LogReg classifier for the E-Shopper use case and ~14 less for the Train-Ticket use case. In addition, we note how the

[1]https://github.com/SpencerLabAQ/icpe-data-challenge-delag
[2]https://github.com/SEALABQualityGroup/E-Shopper

grammar construction time has less variability compared to the training time of the model.

**Answer to RQ1**
Our proposed approach requires a time to construct a grammar that is ∼13 seconds lower compared to the training time of a LogReg classifier for the E-Shopper use case and ∼14 seconds lower for the Train-Ticket use case. Moreover, the time required to build the grammar is almost constant over different runs.

## 4.2 Addressing RQ2

The second research question focuses on the effectiveness of our proposed approach in detecting anomalies compared to a LogReg classifier. To answer this question, we used our approach to detect the anomalies on both E-Shopper and Train Ticket datasets and compared their effectiveness with the LogReg baseline. More in detail, for each dataset, we perform a train-test split and use 80% of the data to build the grammar/train the LogReg model, and we predict anomalies on the remaining 20%. We employ accuracy [24], precision, and recall [7] scores as effectiveness metrics. Concerning the setting of hyper-parameters, for LogReg, we used the default ones provided by the `scikit-learn` library. Instead, the only hyper-parameter required by our approach is the number of bins used by the SAX encoder algorithm [25], which we set to 5. We tested different values of this parameter and found that 5 achieves the highest effectiveness in both use cases.

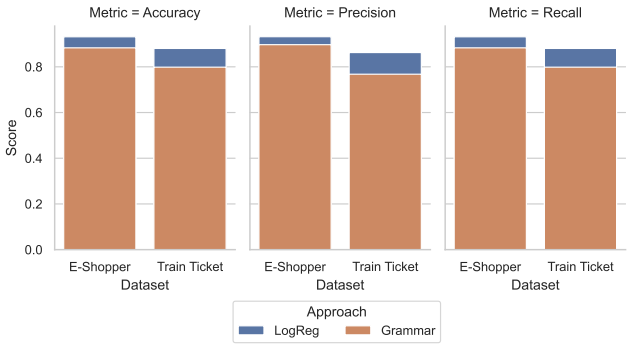



**Figure 3: Comparison of accuracy, precision and recall scores of Logistic Regression and our grammar-based approach.**

Figure 3 reports the accuracy, precision and recall scores for E-Shopper and Train Ticket use cases. The blue bar shows the results achieved by LogReg while the orange bar displays the results achieved by our grammar-based approach. As can be seen, our approach has an effectiveness that is almost comparable to the one achieved by the LogReg model, with a difference of, at most, 10% in the precision score for the Train Ticket dataset. However, we also observe how, in general, the LogReg classifier has a lower effectiveness in the Train Ticket dataset, meaning that the anomaly patterns are less evident in this use case.

**Answer to RQ2**
Our approach has an effectiveness that is almost comparable to the one achieved by a LogReg classifier, with a delta of at most 10% in a use case with less evident anomaly patterns.

## 4.3 Discussion

The performed experiments showed how our proposed approach achieves a higher efficiency in terms of time required to construct the grammar, with a little cost in terms of prediction effectiveness compared to a Logistic Regression classifier. However, it is worth noticing how the efficiency and effectiveness of our approach are also related to the algorithms employed for SAX encoding and grammar induction. Concerning the SAX encoding, in this preliminary work, we employ an implementation of the classical algorithm proposed in [25]. However, other versions of the algorithm have been proposed in the literature that may better detect the differences in anomaly execution traces [20, 27, 30].

The same can be said for the grammar induction algorithm. In this work, we employ the Sequitur algorithm which is one of the most adopted algorithms for grammar induction. However, the grammars generated by this algorithm are often too large and not optimal. Finding the minimum grammar representing a specific language is known to be an NP-complete problem [12]. Nevertheless, some works have been proposed in the last years that try to achieve this task [3, 15].

# 5 CONCLUSION AND FUTURE WORK

In this paper, we introduced an innovative method for anomaly detection utilizing Context-free Grammar, serving as an alternative to Machine Learning techniques. We formally outlined our methodology and introduced an initial implementation using a naive grammar induction algorithm. We then presented preliminary results, which demonstrated comparable effectiveness to Logistic Regression with minimal training time requirements. The presented methodology is adaptable and can be easily tuned based on domain-specific parameters. In addition, the grammar induction algorithm can be interchangeable based on specific needs. Future work avenues include exploring more sophisticated SAX encoding algorithms like the ones proposed in [20, 27, 30], along with an automatic approach for the selection of $\Sigma$. Next, we plan to investigate other grammar induction algorithms like ARVADA [15], which can produce more readable and shorter context-free grammars. Another possible approach would be the hand-crafting of regular expressions or grammars by domain experts. Our methodology can also be easily expanded to account for explainability needs. By constructing the parse tree of a given string, we can easily visualize where the anomaly happened and what microservices it encompassed.

# ACKNOWLEDGMENTS

Part of the numerical simulations have been realized on the Linux HPC cluster Caliban of the High-Performance Computing Laboratory of the Department of Information Engineering, Computer Science and Mathematics (DISIM) at the University of L'Aquila.

# REFERENCES


[1] Nuha Alshuqayran, Nour Ali, and Roger Evans. 2016. A systematic mapping study in microservice architecture. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 44–51.

[2] Mohamed Aly. 2005. Survey on multiclass classification methods. *Neural Netw* 19, 1 (2005), 9. Publisher: Citeseer.

[3] Mohammad Rifat Arefin, Suraj Shetiya, Zili Wang, and Christoph Csallner. 2024. Fast Deterministic Black-box Context-free Grammar Inference. arXiv:2308.06163 [cs.SE]

[4] Chetan Bansal, Sundararajan Renganathan, Ashima Asudani, Olivier Midy, and Mathru Janakiraman. 2020. DeCaf: diagnosing and triaging performance issues in large-scale cloud services. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice* (Seoul, South Korea) *(ICSE-SEIP '20)*. Association for Computing Machinery, New York, NY, USA, 201–210. https://doi.org/10.1145/3377813.3381353

[5] Steven Bird. 2006. NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*. 69–72.

[6] Konstantinos Bountrogiannis, George Tzagkarakis, and Panagiotis Tsakalides. 2021. Anomaly Detection for Symbolic Time Series Representations of Reduced Dimensionality. In *2020 28th European Signal Processing Conference (EUSIPCO)*. 2398–2402. https://doi.org/10.23919/Eusipco47968.2020.9287474

[7] Michael Buckland and Fredric Gey. 1994. The relationship between recall and precision. *Journal of the American society for information science* 45, 1 (1994), 12–19. Publisher: Wiley Online Library.

[8] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. 2019. Machine learning interpretability: A survey on methods and metrics. *Electronics* 8, 8 (2019), 832.

[9] Andrea D'Angelo and Giordano d'Aloisio. 2024. *Grammar-Based Anomaly Detection of Microservice Systems Execution Traces Replication Package*. https://doi.org/10.5281/zenodo.10806012

[10] Qingfeng Du, Tiandi Xie, and Yu He. 2018. Anomaly detection and diagnosis for container-based microservices with performance monitoring. In *Algorithms and Architectures for Parallel Processing: 18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part IV 18*. Springer, 560–572.

[11] Raphael Fischer, Matthias Jakobs, Sascha Mücke, and Katharina Morik. 2022. A Unified Framework for Assessing Energy Efficiency of Machine Learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 39–54.

[12] E Mark Gold. 1978. Complexity of automaton identification from given data. *Information and Control* 37, 3 (1978), 302–320. https://doi.org/10.1016/S0019-9958(78)90562-4

[13] Fabio Guigou, Pierre Collet, and Pierre Parrend. 2019. SCHEDA: Lightweight euclidean-like heuristics for anomaly detection in periodic time series. *Applied Soft Computing* 82 (2019), 105594. https://doi.org/10.1016/j.asoc.2019.105594

[14] Mingxu Jin, Aoran Lv, Yuanpeng Zhu, Zijiang Wen, Yubin Zhong, Zexin Zhao, Jiang Wu, Hejie Li, Hanheng He, and Fengyi Chen. 2020. An Anomaly Detection Algorithm for Microservice Architecture Based on Robust Principal Component Analysis. *IEEE Access* 8 (2020), 226397–226408. https://doi.org/10.1109/ACCESS.2020.3044610

[15] Neil Kulkarni, Caroline Lemieux, and Koushik Sen. 2021. Learning Highly Recursive Input Grammars. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 456–467. https://doi.org/10.1109/ASE51524.2021.9678879

[16] Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. 2022. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering* 27 (2022), 1–28.

[17] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. 2000. A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms. *Machine Learning* 40, 3 (Sept. 2000), 203–228. https://doi.org/10.1023/A:1007608224229

[18] Francesca Marzi, Giordano d'Aloisio, Antinisca Di Marco, and Giovanni Stilo. 2023. Towards a Prediction of Machine Learning Training Time to Support Continuous Learning Systems Development. *arXiv preprint arXiv:2309.11226* (2023).

[19] Scott Menard. 2002. *Applied logistic regression analysis*. Vol. 106. Sage.

[20] Muhammad Marwan Muhammad Fuad. 2012. Genetic algorithms-based symbolic aggregate approximation. In *Data Warehousing and Knowledge Discovery: 14th International Conference, DaWaK 2012, Vienna, Austria, September 3-6, 2012. Proceedings 14*. Springer, 105–116.

[21] Nadia Nahar, Haoran Zhang, Grace Lewis, Shurui Zhou, and Christian Kästner. 2023. A Meta-Summary of Challenges in Building Products with ML Components – Collecting Experiences from 4758+ Practitioners. https://doi.org/10.48550/arXiv.2304.00078 arXiv:2304.00078 [cs].

[22] Craig G Nevill-Manning and Ian H Witten. 1997. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research* 7 (1997), 67–82.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[24] G.H. Rosenfield and K. Fitzpatrick-Lins. 1986. A coefficient of agreement as a measure of thematic classification accuracy. *Photogrammetric Engineering and Remote Sensing* 52, 2 (1986), 223–227. http://pubs.er.usgs.gov/publication/70014667

[25] Hagit Shatkay and Stanley B Zdonik. 1996. Approximate queries and representations for large data sequences. In *Proceedings of the Twelfth International Conference on Data Engineering*. IEEE, 536–545.

[26] Jacopo Soldani and Antonio Brogi. 2022. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *ACM Comput. Surv.* 55, 3, Article 59 (feb 2022), 39 pages. https://doi.org/10.1145/3501297

[27] Youqiang Sun, Jiuyong Li, Jixue Liu, Bingyu Sun, and Christopher Chow. 2014. An improvement of symbolic aggregate approximation distance measure for time series. *Neurocomputing* 138 (2014), 189–198.

[28] Luca Traini and Vittorio Cortellessa. 2023. DeLag: Using Multi-Objective Optimization to Enhance the Detection of Latency Degradation Patterns in Service-Based Systems. *IEEE Transactions on Software Engineering* 49, 6 (2023), 3554–3580. https://doi.org/10.1109/TSE.2023.3266041

[29] Li Wu, Jasmin Bogatinovski, Sasho Nedelkoski, Johan Tordsson, and Odej Kao. 2020. Performance diagnosis in cloud microservices using deep learning. In *International Conference on Service-Oriented Computing*. Springer, 85–96.

[30] Yufeng Yu, Yuelong Zhu, Dingsheng Wan, Huan Liu, and Qun Zhao. 2019. A novel symbolic aggregate approximation for time series. In *Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication (IMCOM) 2019 13*. Springer, 805–822.

# Analyzing Performance Variability in Alibaba's Microservice Architecture: A Critical-Path-Based Perspective

Alireza Ezaz
Brock University
St. Catharines, Ontario, Canada
sezaz@brocku.ca

Ghazal Khodabandeh
Brock University
St. Catharines, Ontario, Canada
gkhodobandeh@brocku.ca

Naser Ezzati-Jivan
Brock University
St. Catharines, Ontario, Canada
nezzatijivan@brocku.ca

## ABSTRACT

In large-scale microservice architectures, such as those utilized by Alibaba, identifying and addressing performance bottlenecks is a significant challenge due to the complicated interactions between thousands of services. To navigate this challenge, we have developed a critical-path-based technique aimed at analyzing microservice interactions within these complex systems. This technique facilitates the identification of critical nodes where service requests experience the longest delays. Our contribution is the discovery of performance variability in service interactions' response times within these critical paths, and pinpointing specific interactions within the system that show a high degree of performance variability. This improves the ability to detect service performance issues and their root causes allowing for dynamic adjustment in data collection detail, and targets critical interactions for adaptive monitoring.

## CCS CONCEPTS

• **General and reference** → **Performance**; • **Software and its engineering** → *Software maintenance tools*; • **Applied computing** → **Service-oriented architectures**.

## KEYWORDS

Microservice Architecture, Performance Bottlenecks, Critical Path, Response Time Variability, Adaptive Tracing

**ACM Reference Format:**
Alireza Ezaz, Ghazal Khodabandeh, and Naser Ezzati-Jivan. 2024. Analyzing Performance Variability in Alibaba's Microservice Architecture: A Critical-Path-Based Perspective. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3651845

## 1 INTRODUCTION

Distributed tracing is crucial for tracking how applications perform across a system, ensuring that operations are smooth and reliable. Unlike tools that focus on individual components, distributed tracing monitors entire requests as they traverse various parts of the system architecture, from a user's click to data storage in a database [9]. This monitoring plays an important role in identifying where issues begin, spotting any behavior that does not align with the expected operation of the system, detecting deviations in performance and finally enhancing the system's overall effectiveness.

However, the challenge intensifies when managing large-scale microservice-based applications like those at Alibaba, Uber, and Amazon, where thousands of services are constantly interacting. The complexity of these systems increases by complicated dependencies among services, the large number of monitoring metrics (e.g., Netflix exposes 2 million metrics and Uber exposes 500 million metrics [10]), and frequent updates, all alongside the extensive data produced by distributed tracing [11]. These factors make it particularly difficult to diagnose performance issues and pinpoint their root causes. To effectively manage these challenges, accurate detection of performance issues and clever analysis are essential.

Recent studies such as [6, 7, 12] have highlighted advancements in microservices monitoring and analysis, employing scalable, real-time frameworks and delving into microservice dependencies and performance. These contributions emphasize the utility of machine learning in predicting usage patterns and the importance of understanding microservice dependencies for improved performance analysis. However, a critical gap remains in the analysis of performance variability across microservices as those approaches often overlook the insights that can be gained from this analysis across different system components.

This is where we take performance variation analysis into account. By looking at how performance values fluctuate, we can find patterns or issues that point to the root of the problems. Consider a scenario in which a single microservice is involved in handling (a part of) three requests within a span of three minutes, but its latency varies significantly for each request. The first request is processed in just 1 millisecond, the second takes longer, at 30 milliseconds, and the third request experiences a substantial delay, requiring 500 milliseconds to complete. Such a notable difference in the service time of the same microservice, known as performance variability, if persistent across numerous interactions, is a red flag that indicates a potential problem warranting further investigation.

Building on this idea, our contribution is conducting a deeper analysis of the concept of performance variability in a large-scale microservice architecture. In our proposed technique, we begin with extracting critical paths from the trace dataset. A critical path is defined as a sequence of service interactions where requests experience the longest delays. After extracting critical paths from the requests, we group them into categories based on how similar they are to each other. For each group of requests sharing the same critical path, our technique focuses on extracting microservice

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651845

interactions within these paths. It involves collecting response times for each interaction from the caller (upper) microservice to the callee (downstream) service over fixed time intervals. Then we calculate average response times and their variance for these interactions. By examining the collected data, our technique aims to identify abnormally high or low response time variations. This step is crucial for pinpointing specific microservices that deviate from expected performance norms, suggesting potential areas of concern. Such analysis uncovers deeper issues, from resource constraints to inefficient microservice executions, offering a refined approach to diagnosing and optimizing microservice issues.

## 2 RELATED WORK

Past efforts introduce various frameworks and techniques for monitoring microservice architecture and analysis of dependencies by creating call graphs and evaluating them for anomaly detection, root cause analysis, and further system optimization. However, they often overlook the insights that can be gained from performance variability analysis. Luo et al. [6] present a comprehensive study of large-scale microservice deployments in AliBaba's production clusters, focusing on the structural properties of microservice call graphs and call dependencies. It also offers an in-depth characterization of microservice runtime performance, providing insights into scheduling and resource management. Barham et al. [3] introduce Magpie, a tool designed to model and analyze system workloads by capturing resource consumption and control paths of requests in a system. It features an approach for detailed workload characterization without requiring modifications to the system, enabling accurate performance analysis and debugging. Similarly, Thalheim et al. [10] designed Sieve to derive actionable insights from monitored metrics in distributed systems. Sieve features a metrics reduction framework and a metrics dependency extractor, which together help in filtering out unimportant metrics and inferring metrics dependencies, thereby enhancing the management of microservices-based applications analysis. Other studies combine machine learning techniques with their designed frameworks to find interesting results in microservice systems. chen et al. [4] introduce a deep learning approach to automate fault diagnosis with high precision, to detect faults, and identify root causes effectively. In another study, Janecek et al. [5] introduced a framework for detecting performance anomalies in software systems through the analysis of system-level trace data by employing critical path analysis and machine learning clustering. Nevertheless, these studies focus on anomaly detection and system monitoring, bypassing an in-depth exploration of performance variability and its implications for system optimization.

## 3 METHODOLOGY

Building upon the foundational work of Ates et al. [2] and Sambasivan et al. [8], our methodology leverages the premise that performance variation is indicative of unknown system behaviors and that requests following similar workflows are expected to yield similar runtime footprints. Our study utilizes data from Alibaba's production clusters[1] [1], focusing specifically on the first one hour of the dataset. This initial period encompasses traces across nearly twenty thousand microservices, offering a concise yet significant observation window. The data includes service IDs within the call graph, performance metrics such as response time, and the relationships between upstream (caller) and downstream (callee) microservices, preparing the basis for our analysis.

Our proposed methodology, as shown in Fig 1, involves data preprocessing, critical path extraction, performance variation analysis within the critical paths, and visualization tools. Our developed tool and scripts used in this methodology are available online at [2]. In the subsequent sections, we will elaborate on each step separately.

### 3.1 Data Collection and Preprocessing

As shown in stage (1) of Fig 1, our method begins with the extraction and preprocessing of the trace data. The preprocessing phase is important for ensuring data integrity and usability. The following steps outline our preprocessing approach:

(1) **Dataset Cleaning:** We cleaned the dataset by removing entries with invalid or non-numeric values for response times.
(2) **Invalid Trace ID Removal:** We identified and excluded records associated with invalid response time values by isolating unique trace IDs corresponding to these records.
(3) **Null Value Handling:** Further cleaning involved discarding records containing any empty fields.
(4) **Unnecessary Features Removal:** We also removed unnecessary features to focus the dataset on essential metrics relevant to our study such as timestamp, trace ID, um (upper microservice), dm (downstream microservice), rt (response time of dm to um).

### 3.2 Critical Path Extraction

Following the preprocessing of Alibaba's microservice interaction data, our methodology advances to stage (2) in Fig 1; extraction of critical paths. The extraction process is detailed as follows:

(1) **Critical Path Identification:**
At this stage, we identify each request in AliBaba's trace dataset by filtering unique trace IDs. For each request, records are sorted by timestamp to establish the sequence of interactions. The 'endtime' for each interaction is calculated by adding its response time to its timestamp. The interaction with the longest 'endtime' signifies the end of the critical path. Tracing back from this endpoint, the sequence of services involved in the critical path, from upstream to downstream, is determined.
(2) **Grouping Requests Based on Critical Path Similarity:** Under the assumption that requests with similar workflows should yield similar performance metrics [2], our designed technique groups requests sharing an exact similar critical path.

By focusing on these critical paths extracted , we aim to identify the key areas where performance optimization efforts should be concentrated.

[1] https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2022

[2] https://github.com/Alireza-Ezaz/Analyzing-Performance-Variability-in-Alibaba-s-Microservice-Architecture

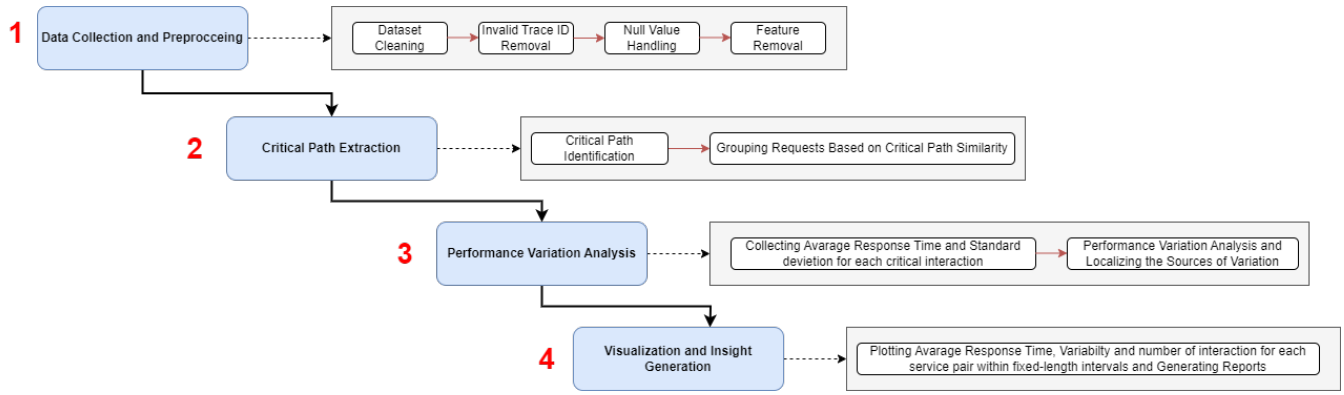



**Figure 1: The Architecture of Our Proposed Technique**

## 3.3 Performance Variation Analysis

Moving on to stage (3) in Fig 1, we investigate the performance variation within each group of requests with the same critical path. We then focus on how each specific microservice involved in the path, exhibit variability in its response times.

*3.3.1 Collecting Average Response Time and Standard deviation for each critical interaction.* For groups of requests following the same critical path, our method records the response times of each microservice interaction within these paths. This is done over twenty 3-minutes intervals creating a 1-hour observation window. After collecting these response times, it calculates the average response times and standard deviations, for each of the interactions, aiming to further localize the performance issue into a specific critical interaction in the next step.

*3.3.2 Performance Variation Analysis and Localizing the Sources of Variation.* As the core of our contribution, We analyze the variability in response times for each interaction within the extracted critical paths. Having the average and standard deviation for each interaction from the previous step, the average provides a baseline of expected performance, while the standard deviation reveals the extent of variability, offering insights into the consistency of the corresponding microservice's response. This dual metric approach enables a more granular understanding of performance variations, uncovering more details about microservice interactions. High variability (a large standard deviation compared to the average response time) suggests potential areas of concern, signaling that the performance of certain interactions (critical interactions) within the critical path is inconsistent and the problem can be localized to those specific critical interactions. This inconsistency may be indicative of deeper issues such as network latency, resource contention, or issues in service dependencies, which could adversely affect the overall system performance.

## 3.4 Visualization and Insight Generation

For further analysis in stage (4) of Fig 1, we plot the average response times and standard deviations across predetermined three-minute intervals, highlighting interactions that exhibit significant variability. These plots are helpful in pinpointing areas of instability, suggesting where further optimization and investigation are necessary. For each interaction characterized by high variability (our criteria for 'high'—selects interactions where the standard deviation exceeds ten times the mean response time), we generate dual-axis plots that represent the average response times against the occurrence counts over the intervals. At the end, we generate a report summarizing insights, including the number of unique traces examined, the variety of critical paths identified, and interactions with notable performance variability. This summary, alongside detailed statistics and visualizations, is compiled into user-friendly formats, ensuring that the insights are accessible to stakeholders involved in system development and optimization.

# 4 ANALYSIS AND DISCUSSION

Our investigation into the performance variation within Alibaba's microservice architecture led to the extraction of 91,704 unique critical paths from a 1-hour interval dataset of 40,062,862 requests, each distinguished by a unique trace ID. Of these, 1,891 microservice interactions within critical paths exhibited a noticeably high variance in performance. Our analysis has unfolded insightful patterns of response time variability. Delving into the data, we have presented twelve selected plots in Fig 2, each uncovering a critical interaction within the corresponding critical path. Our analysis is based on interactions grouped into four general patterns, characterized by abnormal fluctuations within at least one three-minute interval in an hour.

The analysis of data through our technique is visually represented in Fig 2, where the x-axis segments the observation window into 3-minute intervals. Green bars graphically depict the average response time per interval, with numerical standard deviation values indicated above, offering a clear view of performance fluctuations. The blue line complements this by illustrating interaction counts, and black error bars extend from each green bar to represent the range of response time variability, providing an overview of the interaction's performance variability. The presence of error bars extending into negative values does not imply negative response times; they simply show that the lower range of variability falls below the mean due to the subtraction of the standard deviation from the average. Interaction counts should be interpreted with reference to the right y-axis, and average response times should be related to the left y-axis.

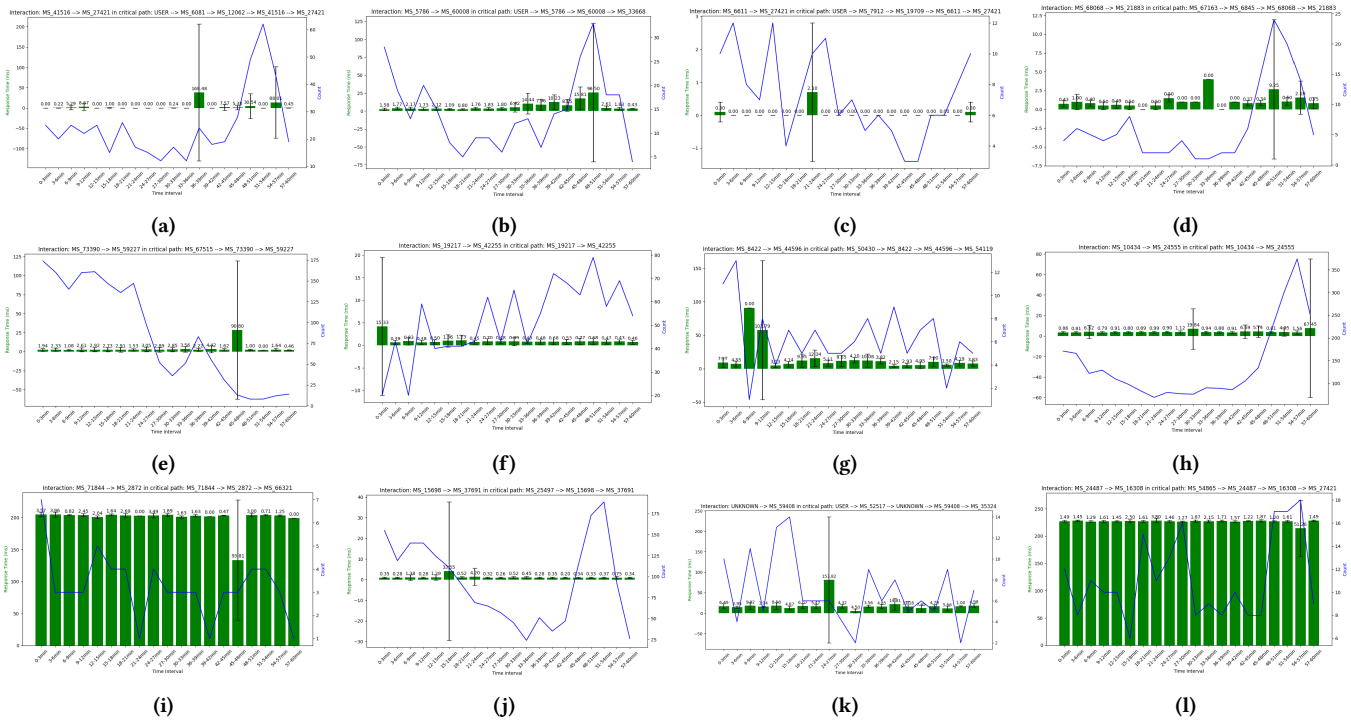


**Figure 2: Performance Variability Analysis for Different Critical Interactions within Corresponding Critical Paths**

Among the patterns observed, one notable scenario in Figures 2a, 2b, 2c, and 2d showcased some periods where the interaction count, average response time, and the variation in response time were simultaneously high. This indicates a potential overload scenario, suggesting the system was handling a higher volume of requests

Conversely, we also identified intervals like the first 30 minutes of Fig 2e where, despite a high number of interactions, the average response time and standard deviations remained stable, hinting at the system's ability to efficiently manage load without compromising on performance consistency.

Another pattern such as 36-39 minute interval in Fig 2a, 45-48 minute interval in Fig 2e, 0-3 minute interval in Fig 2f, 30-33 minute interval in Fig 2g, and 0-3 minute interval in Fig 2h, highlighted through our analysis was the occurrence of intervals where an increase in the average response time was accompanied by substantial variability, despite a lower interaction count compared to peak periods. This indicates that factors other than the volume of interactions, such as resource allocation or network issues, could be impacting performance.

Furthermore, we observed a critical interaction in 45-48 minute interval in Fig 2i where the average response time either decreased or remained low, yet the variability in response times increased. This suggests a growing inconsistency in how requests are processed, with some being completed swiftly while others face delays, leading to an unpredictable performance landscape.

In Figures 2j, 2k, and 2l we see a combination of the above scenarios of system performance previously discussed, each highlighting different ways that performance can vary under various conditions. Its analysis emphasizes the complex nature of performance issues, showing how different factors can impact the system's effectiveness and dependability. By analyzing these figures, we get a deeper insight into the system's behavior, which helps identify specific areas (critical interactions) that could benefit from optimization or further detailed study.

Overall, our results indicate that our approach provides a targeted method for monitoring and diagnosing the system's behavior and directs developers toward potential points of optimization, thereby mitigating system failures or inefficiencies. It also identifies critical interactions with high performance variation as prime candidates that can be considered for adaptive tracing and monitoring.

## 5 CONCLUSIONS AND FUTURE WORK

We analyzed performance variations within Alibaba's microservice architecture, focusing on identifying critical paths and analyzing response time variability. Our findings emphasize the complexity of managing performance in microservice architectures and the importance of continuous monitoring and analysis to identify and address bottlenecks.

Future work will extend this analysis by incorporating additional performance metrics such as CPU and memory utilization, applying machine learning algorithms to predict potential performance bottlenecks, and enhancing trace grouping techniques to efficiently manage and analyze large datasets. These efforts aim to improve the understanding and management of system performance, leading to more robust and efficient microservice architectures.

# REFERENCES


[1] Alibaba Group. 2022. Alibaba Cluster Data - Cluster Trace of Microservices. https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2022. Accessed: YYYY-MM-DD.

[2] Emre Ates, Lily Sturmann, Mert Toslali, Orran Krieger, Richard Megginson, Ayse K Coskun, and Raja R Sambasivan. 2019. An automated, cross-layer instrumentation framework for diagnosing performance problems in distributed applications. In *Proceedings of the ACM Symposium on Cloud Computing*. 165–170.

[3] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. 2004. Using Magpie for request extraction and workload modelling.. In *OSDI*, Vol. 4. 18–18.

[4] Hao Chen, Kegang Wei, An Li, Tao Wang, and Wenbo Zhang. 2021. Trace-based intelligent fault diagnosis for microservices with deep learning. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 884–893.

[5] Madeline Janecek, Naser Ezzati-Jivan, and Seyed Vahid Azhari. 2021. Container workload characterization through host system tracing. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 9–19.

[6] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*. 412–426.

[7] Barakat Saman. 2017. Monitoring and analysis of microservices performance. *Journal of Computer Science and Control Systems* 10, 1 (2017), 19.

[8] Raja R Sambasivan and Gregory R Ganger. 2012. Automated diagnosis without predictability is a recipe for failure. In *4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 12)*.

[9] Yuri Shkuro. 2019. *Mastering Distributed Tracing: Analyzing performance in microservices and complex systems*. Packt Publishing Ltd.

[10] Jörg Thalheim, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. 2017. Sieve: Actionable insights from monitored metrics in distributed systems. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. 14–27.

[11] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. 2020. Microrca: Root cause localization of performance issues in microservices. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–9.

[12] Cathy H Zhang and M Omair Shafiq. 2022. A Real-time, Scalable Monitoring and User Analytics Solution for Microservices-based Software Applications. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 6125–6134.

# LLaMPS: Large Language Models Placement System

Likhith Bandamudi
TCS Research
likhith.bandamudi1@tcs.com

Ravi Kumar Singh
TCS Research
ravik.singh2@tcs.com

Shruti Kunde
TCS Research
shruti.kunde@tcs.com

Mayank Mishra
TCS Research
mishra.m@tcs.com

Rekha Singhal
TCS Research
rekha.singhal@tcs.com

## ABSTRACT

The rapid expansion of Large Language Models (LLMs) presents significant challenges in efficient deployment for inference tasks, primarily due to their substantial memory and computational resource requirements. Many enterprises possess a variety of computing resources—servers, VMs, PCs, laptops—that cannot individually host a complete LLM. Collectively, however, these resources may be adequate for even the most demanding LLMs.

We introduce LLaMPS, a novel tool, designed to optimally distribute blocks [1] of LLMs across available computing resources within an enterprise. LLaMPS leverages the unused capacities of these machines, allowing for the decentralized hosting of LLMs. This tool enables users to contribute their machine's resources to a shared pool, facilitating others within the network to access and utilize these resources for inference tasks. At its core, LLaMPS employs a sophisticated distributed framework to allocate transformer blocks of LLMs across various servers. In cases where a model is predeployed, users can directly access inference results (GUI and API). Our tool has undergone extensive testing with several open-source LLMs, including BLOOM-560m, BLOOM-3b, BLOOM-7b1, Falcon-40b, and LLaMA-70b. It is currently implemented in a real-world enterprise network setting, demonstrating its practical applicability and effectiveness.

## CCS CONCEPTS

• **Computing methodologies** → **Distributed AI Tool**.

## KEYWORDS

LLMs, Distributed Inference, Optimal block placement

**ACM Reference Format:**
Likhith Bandamudi, Ravi Kumar Singh, Shruti Kunde, Mayank Mishra, and Rekha Singhal. 2024. LLaMPS: Large Language Models Placement System. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3629527.3651404

[1]Large Language models contain multiple transformer blocks which can be distributed across machines.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651404

## 1 INTRODUCTION

Large Language Models (LLMs) have become pervasive and now play a crucial role in business operations by facilitating customer interactions and offering recommendations. As the LLMs continue to grow in complexity, their size expands proportionally (with an increase in parameters), necessitating increased computational capacity and memory to ensure optimal functional behavior. This is a significant challenge for businesses with limited computational resources, particularly in enterprises where numerous individual devices/resources may lack the capability to independently host an entire LLM. However, a viable alternative emerges as businesses

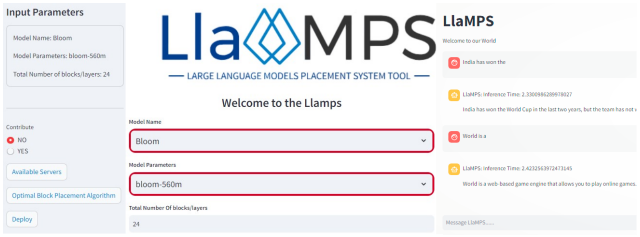



**Figure 1: User interface of LLaMPS**

consider leveraging the collective capabilities of multiple computers resembling a collaborative effort to handle substantial workloads. This approach explores the concept of "leftover" capacity within large enterprises, where individual devices/resources may not possess the capability to independently host an LLM. By distributing the computational load across multiple machines throughout an enterprise, businesses can effectively utilize their leftover capacity, enabling the deployment of LLMs in a distributed manner. This innovative approach introduces new possibilities for inference and other downstream tasks.

There exist some works in the literature such as Petals, DeepSpeed, and Zero Inference which facilitate distributed inference. However, none of them explore the prospect of utilizing *leftover* capacity already available in an enterprise to deploy LLMs. Also, they do not optimally distribute transformer blocks such that the number of clients served, are maximized; or that multiple LLMs can be optimally deployed using available capacity. We proposed the OPA Optimal Placement Algorithm (details in [1]). We have built a tool, LLaMPS, which facilitates the optimal placement of transformer blocks on distributed resources in an enterprise-wide network. LLaMPS utilizes Petals as the underlying distributed framework in its current implementation. However, it is agnostic to the underlying framework and is not bound by it. LLaMPS has the following features:

- **Leftover resource utilization:** Enables collaborative sharing of leftover capacity across multiple devices within the enterprise.
- **Optimal block distribution:** Ensures optimal distribution of transformer blocks on an enterprise-wide scale, using OPA (optimal placement algorithm).
- **Multi-Client and Multi-model support:** Supports multiple client requests concurrently. Enables loading of blocks associated with multiple LLMs on a single resource, thus optimizing overall resource usage in the enterprise.
- **Cost optimization:**LLaMPS distributes LLM across multiple devices, utilizing leftover resources, reducing the need for expensive hosting.

LLaMPS is a tool that adopts a client-server architecture, developed in Python, and utilizes the underlying open-source Petals distributed framework. It has a web-based interface, featuring a user-friendly Streamlit-built GUI. LLaMPS is adaptable and offers flexibility to support future alternative distributed frameworks.

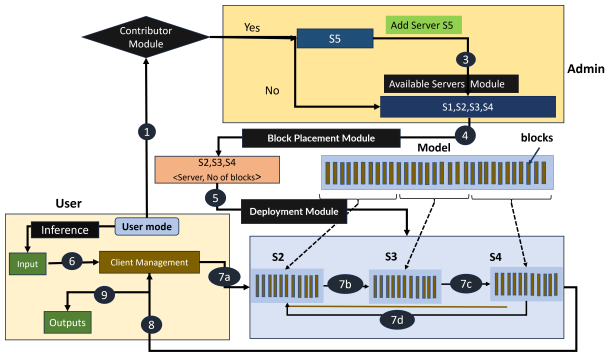



**Figure 2: LLaMPS Architecture**

## 2 LLAMPS ARCHITECTURE

We now discuss the architecture of LLaMPS depicted in Figure 2.

The user interacts with the LLaMPS tool via the user interface (Figure 1). The **Contributor Module** enables the user to *share or contribute* leftover capacity of his device with the distributed network. If the user chooses to contribute, the IP address and server information (available memory and cores) of the device are provided to the **Admin**. If not, it implies that the user only wishes to perform an *Inference* task.

**Available Servers Module** enables displaying a list of resources that are available on the network, along with their IP addresses, available memory, and cores on the UI. This information is then sent to the admin.

The **Block Placement Module** then kicks off the OPA (Optimal block Placement Algorithm) which takes input all info about the available servers and creates a plan for block placement. The plan outlines a list of selected servers (from the available servers) and the distribution of blocks to be loaded on each server for efficient resource utilization. Additional details of OPA can be found in our paper [1].

The **Deployment Module** deploys the blocks on the servers as per the plan. For example, Blocks (0:12) are loaded on Server 1; Blocks (12:24) are loaded on Server 2, and so on. Once the blocks are deployed, the user can start using the deployed model for the inference task. The user inputs text and initiates the **Inference process**.

In the client, a route is formed based on the sequence of blocks deployed on multiple servers, and the inference runs across these servers. The inference output from the blocks is then converted into a human-readable form and the output is then transmitted to the user interface.

## 3 USE CASE

Figure 1 depicts the user interface of the LLaMPS tool. The user will input the model name and the number of parameters of the model. The number of blocks corresponding to the model will be automatically displayed. The user may choose to contribute or share his device's leftover resources or perform inference without sharing any resources. Upon clicking the *Available Servers* button, the list of available servers will be displayed. The user then clicks the *Optimal Placement Algorithm* button which creates a plan for optimal distribution of blocks. The blocks are deployed by clicking the *Deploy* button. Once the deployment is complete, the user can then perform inference as displayed in the right pane.

Assume that an enterprise has the following list of available servers s1<155, 8>, s2<113.5, 8>, s3<83.5,2>, s4<83.5,4>, and s5<83.5,8>. The first value in the tuple represents the available leftover memory in GBs and the second value represents the number of available cores. The user wants to deploy the LLama2 70b model, which comprises 80 transformer blocks. The size of LLaMA2 is approximately 300GB including overheads. No single server within the enterprise can host the entire LLama2 model. By leveraging the combined memory capacities of multiple servers, the OPA algorithm of LLaMPS creates a plan for deployment. Servers s1, s2, and s5 are selected taking into account available memory and cores and will host transformer blocks 0-34, 35-60, and 61-79 respectively.

During an inference cycle, the client management(step 7a in Figure 2) tokenizes the input. The tokenized input is then relayed to server s1, where it traverses the allocated transformer blocks. The intermediate output is sequentially passed to server s2 and subsequent servers until it has traversed all the transformer blocks. The final output from the last server in the sequence is transmitted back to the client(step 8 in Figure 2), where the required output is generated.

## 4 CONCLUSION AND FUTURE WORK

LLaMPS is instrumental in addressing the challenges associated with deploying LLMs on limited computational resources. It achieves this by efficiently distributing the computational workload of LLMs across multiple machines, using the OPA algorithm. OPA not only optimizes the utilization of residual computing power but also manages the dynamically varying leftover memory and compute resources optimally. We plan to enhance the capability of the LLaMPS tool by extending it for optimal utilization of enterprise cloud resources when deploying LLMs.

## REFERENCES

[1] Ravi Kumar Singh, Likhit Bandamudi, Shruti Kunde, Mayank Mishra, and Rekha Singhal. 2024. Leftovers for LlaMA. In *International Conference on Performance Engineering(accepted)*. ICPE.

# Into the Fire: Delving into Kubernetes Performance and Scale with Kube-burner

Sai Sindhur Malleni
Red Hat, Inc.
Bengaluru, India
smalleni@redhat.com

Raul Sevilla Canavate
Red Hat, Inc.
Madrid, Spain
rsevilla@redhat.com

Vishnu Challa
Red Hat, Inc.
Raleigh, US, NC
vchalla@redhat.com

## ABSTRACT

This paper introduces Kube-burner[1], an open-source tool for orchestrating performance and scalability testing of Kubernetes[2], with the ability to operate seamlessly across different distributions. We discuss its importance in the cloud native landscape, features and capabilities and delve into its architecture and usage. Additionally, we also present a case study on performance benchmarking using Kube-burner and subsequent analysis to demonstrate its value.

## KEYWORDS

kubernetes, benchmark, workload, pods, containers, metrics, performance testing, scale testing, openshift

**ACM Reference Format:**
Sai Sindhur Malleni, Raul Sevilla Canavate, and Vishnu Challa. 2024. Into the Fire: Delving into Kubernetes Performance and Scale with Kube-burner. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3629527.3651405

## 1 INTRODUCTION

Microservices deployed as containers have become the prominent way of building and delivering software [3]. This rise in adoption of container technologies as well as microservices based architectures has elevated the importance of container orchestration engines like Kubernetes in empowering modern application development and delivery. In that sense, Kubernetes serves as the fundamental building block of cloud native infrastructure.

As cloud native is all about building, deploying and managing applications at scale, the performance and scale of the underlying Kubernetes platform is of essence. Kube-burner, with its versatile capabilities in scaling, creating, deleting, and patching Kubernetes resources as per user defined scenarios, along with its ability to collect and index metrics from the monitoring system and benchmark results plays a crucial role in illuminating the previously obscure aspects of Kubernetes performance. Furthermore, its custom measurements and alerting features notify users when Key Performance Indicators (KPIs) indicate that Service Level Objectives (SLOs) have been breached, adding an extra layer of insight.

[1]https://github.com/kube-burner/kube-burner
[2]https://kubernetes.io

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651405

## 2 ARCHITECTURE

Kube-burner is a CLI based tool written in Golang[3] that provides three major classes of functionality: **benchmark orchestration** - creating, deleting and patching Kubernetes resources at scale; **measurements** - detailed metrics obtained with the help of the Kubernetes API, among others, such as how long it takes for a pod to go from scheduling to ready (also known as podLatency) and **observability** - the ability to collect metrics about the Kubernetes platform under load by scraping a user defined set of metrics from the Prometheus[4] monitoring stack and index them along with the benchmark results into a long term storage like the local File system of the host from which kube-burner is run or an Elasticsearch[5]/OpenSearch[6] endpoint for subsequent retrieval, visualization and comparison. Kube-burner is also capable of extracting certain metadata that defines the configuration of the Kubernetes platform and appending that to the benchmark result data to facilitate regression testing and comparisons between different configurations such as Container Storage Interface (CSI) and Container Network Interface (CNI) plugins or different distributions of Kubernetes that have very nuanced differences as has been done in some past work [1] [2] albeit using different tools. The resource creation, deletion and patching functionality is implemented using client-go, while the Prometheus and Elasticsearch/OpenSearch clients as well as the metadata collection is implemented as part of a common library called go-commons that is imported as a module. The objects created by Kube-burner are rendered using the default Golang's template library.

## 3 BENCHMARK ORCHESTRATION

Kube-burner is available as a binary that can run on Linux, Windows or Darwin based systems across a wide range of CPU architectures including x86_64 and arm64. Kube-burner in its most common usage is invoked by passing a YAML based configuration file to the executable on the CLI. The configuration file contains certain global configuration options such as the endpoints of the Elasticsearch/OpenSearch indexer followed by a list of jobs, with each job having its own set of supported parameters. Each job could create, delete or patch objects at a rate defined by the QPS and Burst parameters within the job. An example job would be one that creates several deployment objects per namespace across several namespaces (determined by the *jobIterations* parameter) and through each deployment creates multiple pods.

Ready-made benchmarks that mimic production workloads are also available txo users to run directly instead of defining their

[3]https://go.dev
[4]https://prometheus.io
[5]https://www.elastic.co
[6]https://opensearch.org

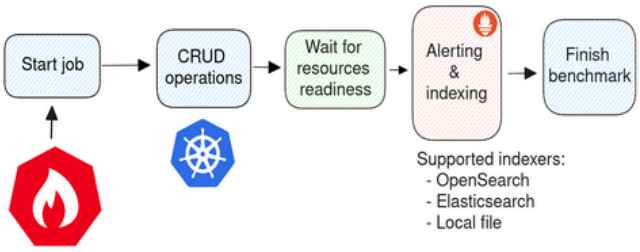



**Figure 1: Kube-burner workflow**

configuration file, providing a trade-off between flexibility and ease of use.

## 4 OBSERVABILITY

Performing a benchmark using Kube-burner is relatively simple. However, it is sometimes necessary to analyze and be able to react to some KPIs in order to validate a benchmark. That is why Kube-burner ships metric-collection and alerting systems based on Prometheus expressions. As stated earlier, Kube-burner is capable of connecting to the Prometheus stack running on the Kubernetes platform and extracting metrics pertaining to the platform's response to the benchmark load. The extracted metrics are then indexed to the configured indexer, either the local File system on the host running kube-burner or an Elasticsearch/OpenSearch endpoint for long term storage, retrieval and further integration with tools capable of graphing data such as Grafana[7]. The metrics collection feature is configured through a file referenced by the *metrics-profile* flag, which can point to a local path or URL of a YAML-formatted file containing a list of the Prometheus expressions that Kube-burner will perform one by one after all the jobs are finished. Kube-burner also includes an alerting feature that is capable of evaluating Prometheus expressions in order to fire and index alerts based on user-specified Prometheus expressions. These alerts could be used to indicate anomalies found in certain Key Performance Indicators.

## 5 MEASUREMENTS

Not all of the data that is capable of providing insights into the performance of the platform during the course of a benchmark run and facilitating debugging once a potential bottleneck has been found can be obtained from the Prometheus stack running on the Kubernetes cluster. For example, there are no readily available metrics in Prometheus to quantify the performance of the platform in terms of the time taken to schedule and run pods during periods of heavy churn on the cluster. Furthermore, there is a need to be able to gather Golang profiling data correlating to the benchmark run from the Kubernetes infrastructure pods such as the API server or etcd. Pprof is another supported custom measurement in Kube-burner which helps the user gather profiling information to further assist advanced debugging. In total, Kube-burner supports three custom measurements during a benchmark run: **podLatency** - for measuring the time it takes for pods to go from scheduling to ready and further reporting quantiles across the entire set of pods created as part of the benchmark, **vmiLatency** - similar to podLatency but for virtual machine instances running on Kubernetes through KubeVirt and **pprof** - for collecting Golang profiling information from Kubernetes infrastructure pods as well as user applications.

[7]https://grafana.com

## 6 CASE STUDY

As part of continuous testing we undertake at Red Hat to establish the performance and scale leadership of OpenShift (Red Hat's enterprise grade Kubernetes distribution), we use Kube-burner extensively to quantify the performance of every release. A relevant recent use case of Kube-burner has been its role in validating the performance and scalability requirements of a CNI plugin in OpenShift (OVNKubernetes) before transitioning to General Availability (GA) and replacing the previous one (OpenShiftSDN) as the default. Workloads and metrics provided by Kube-burner were crucial to detect the different bottlenecks this plugin had. For example, one of these metrics, podLatency, , can provide deep insights into several aspects of a CNI plugin's performance, scalability and efficiency. We used the 99th percentile from the quantiles reported by the podLatency metric to summarize the time taken for the different pod lifecycle stages, starting from their creation and ending up in a ready status, which includes the network setup of the pods. All of this was done during a benchmark that spins up thousands of pods across the cluster. As can be seen from the figure below, the 4.14 release of OpenShift showed improved scalability by being performant at larger cluster sizes.

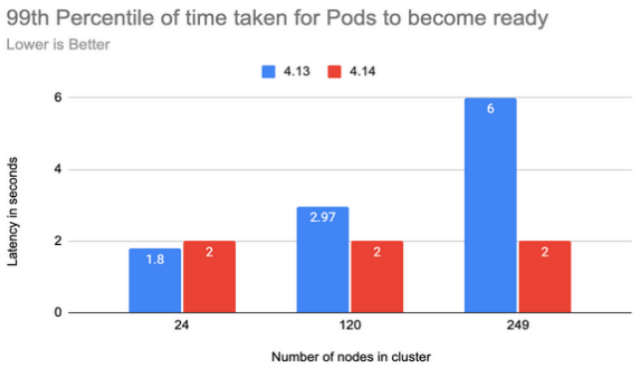



**Figure 2: Measurements from Kube-burner graphed to quantify performance improvements compared to previous release**

## REFERENCES

[1] Heiko Koziolek and Nafise Eskandani. 2023. Lightweight kubernetes distributions: a performance comparison of microk8s, k3s, k0s, and microshift. In ACM/SPEC International Conference on Performance Engineering (ICPE '23), Coimbra, Portugal. ACM, New York, NY, USA. https://doi.org/10.1145/3578244.3583737.

[2] Foutse Khomh Mohab Aly and Soumaya Yacout. 2018. Kubernetes or openshift? which technology best suits eclipse hono iot deployments. In *11th Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 113–120. https://doi.org/10.1109/SOCA.2018.00024.

[3] Olaf Zimmermann. 2017. Microservices tenets. *Computer Science-Research and Development*, 32, 3-4, (July 2017), 301–310. https://doi.org/10.1007/s00450-016-0337-0.

# SuperArch: Optimal Architecture Design for Cloud Deployment

Kuldeep Singh
TCS, India
k.singh13@tcs.com

Chetan Phalak
TCS, India
chetan1.phalak@tcs.com

Dheeraj Chahal
TCS, India
d.chahal@tcs.com

Shruti Kunde
TCS, India
shruti.kunde@tcs.com

Rekha Singhal
TCS, India
rekha.singhal@tcs.com

## ABSTRACT

The success of application migration to cloud depends on multiple factors such as achieving expected performance, optimal cost on deployment, data security etc. The application migration process starts with the architecture design, mapping technical and business specifications to the appropriate services in cloud. However, cloud vendors offer numerous services for each service type and requirement. The onus of selecting the optimal service from the pool lies with the user. Identifying an optimal service for a specific component or application requirement is a daunting task and necessitates a deep understanding of each cloud service offered.

This paper introduces SuperArch, a supervised architecture design tool designed to facilitate optimal selection and configuration of cloud services. We propose utilization of Large Language Models (LLM) for extracting information from user requirements and specifications, aiding in optimal selection of cloud services. Additionally, SuperArch maps workloads to the cloud services to generate optimal configurations of the cloud service and estimate performance and cost of the entire architecture.

## CCS CONCEPTS

• **General and reference** → **Performance**; **Estimation**.

## KEYWORDS

Performance and cost estimation, cloud deployment

**ACM Reference Format:**
Kuldeep Singh, Chetan Phalak, Dheeraj Chahal, Shruti Kunde, and Rekha Singhal. 2024. SuperArch: Optimal Architecture Design for Cloud Deployment. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3629527.3651406

## 1 INTRODUCTION

Creating a resilient and efficient cloud-based system requires a delicate equilibrium among factors such as performance, cost, compliance, security, and user-specific requirements. Navigating the vast and ever-evolving design space encompassing various cloud services, new hardware, and multi-cloud deployments poses a significant challenge for the user. User-provided technical and business requirements may be fulfilled by multiple services from a cloud vendor. For example, an AWS database requirement can be served with Amazon RDS, DynamoDB, Amazon Neptune, or Amazon Aurora. In order to design an optimal architecture for complex user requirements, a conventional approach is based on manually exploiting abilities and experiences of cloud architects and iterative designs. Fortunately, the advent of state-of-the-art LLMs [8] presents an opportunity to find the optimal cloud-based services and their configuration that form the foundation of end-to-end architectures. Currently, the tools available in the market are not fully leveraging the capabilities of Large Language Models (LLMs) to automatically suggest high-performance and cost-effective optimal cloud architectures. This deficiency frequently results in the creation of inefficient designs, leading to lower performance and inaccurate cost estimates, ultimately causing cost escalations. Moreover, the hardware and software components provided by these cloud services can be configured in various ways influencing their performance and cost. A crucial factor in choosing the right cloud service and configuring it appropriately, is the anticipated workload, which is defined by factors such as expected number of users, size of data etc.

| Tool | Cost | | Guided design | Performance | | All * vendors supported | LLM * support |
|---|---|---|---|---|---|---|---|
| | Selected services | End-to-end | | Selected services | End-to-end | | |
| Holori [6] | P | N | N | N | N | Y | N |
| Diagrams.net [4] | N | N | N | N | N | Y | P |
| Cloudskew [3] | N | N | N | N | N | Y | N |
| Cloudcraft [2] | P | P | N | N | N | N | N |
| Hava.io [5] | Y | Y | N | N | N | Y | N |
| Lucidchart [7] | Y | Y | N | N | N | Y | Y |
| Brainboard [1] | Y | Y | N | N | N | Y | N |
| SuperArch | Y | Y | Y | Y | Y | P | Y |

**Table 1: Comparison of SuperArch with cloud architecture design tools**

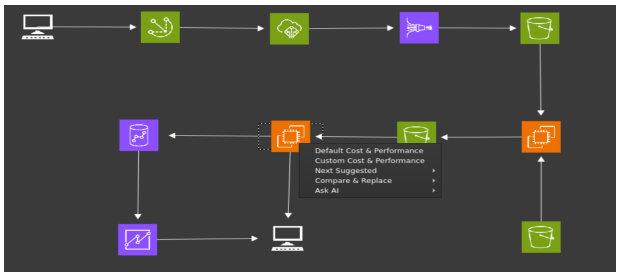
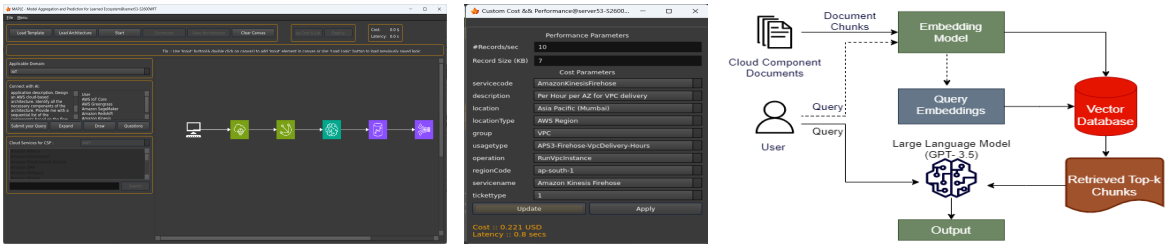



**Figure 1: An example AWS cloud architecture design for IoT application using SuperArch**

Most of the existing architecture design tool (see Table 1) do not map current as well as futuristic workloads to the cloud services in an architecture design often resulting in an inaccurate performance and cost estimation results in cost escalation and hence cloud repatriation.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651406




(a) (b) (c)

**Figure 2: (a) Use of LLaMA for automated architecture design based on user specifications. (b) Cloud deployment cost estimation for AWS S3 service. (c) RAG design for finding architecture patterns from user specifications and vendor documents**

This work aims to address these challenges and contribute to the development of a more nuanced approach to cloud-based system design using a tool called SuperArch (see Figure 1).

## 2 FEATURES OF SUPERARCH

Major contributions of our tool are as follows :

- **Leveraging Language Models:** The tool utilizes LLM to automatically identify and recommend cloud services based on capabilities, features, and system requirements. Additionally, it identifies optimal cloud services by recognizing architecture patterns in user's technical and business specifications. The tool automatically parses cloud services and their connections to draw architecture on the canvas as depicted in Figure 2(a).
- **Dynamic Architecture Modification**: The architecture generated by the LLM is editable which enables iterative modification of the architecture, allowing users to change and compare components at each step, for instance if user wants to compare RDS and DynamoDB it can be done within a few clicks.
- **Cost and Performance Estimation**: A repository of performance benchmarking data from various sources is maintained for popular cloud services such as storage, databases, VMs of various configurations, etc. Also, baseline cost numbers with associated parameters (geography, cost/hour, configuration etc.) are retrieved using cost calculators provided by vendors. This data is used in conjunction with workload data available in the user specification document to calculate end-to-end performance and cost of the architecture as illustrated in Figure 2(b).
- **User-Friendly Interface and multi-cloud support**: Offers an intuitive and user-friendly interface for seamless interaction, and also simplifies the architecture design process by recommending *next optimal cloud service and configuration* at each step in the design process.
- **Iterative Refinement**: Allows iterative refinement of the architecture based on user feedback, ensuring collaborative efforts between the tool's capabilities and user domain expertise. Tool provides features to save and load versions of the architecture.
- **Domain Based Templates:** The tool offers domain-specific templates, tailoring designs to applications of various industries or fields. Users can easily access and utilize pre-designed templates relevant to their application domain, streamlining the designing process and ensuring a professional and industry-appropriate architecture for their application.

## 3 USE CASE

Figure 1 shows an AWS architecture for an IoT application using SuperArch. User specifications for the application requires streaming on-premise data to cloud for storage, model training, performing analytics and sending results back to the user. The architecture is generated using LLM and supervised design functionalities providing assistance to architects in making optimal choices for cloud components from a vast array of available options.

## 4 CONCLUSION AND FUTURE WORK

We proposed the SuperArch tool that utilizes the capabilities of Large Language Models (LLMs) for optimal cloud architecture design. Furthermore, we presented a methodology for estimating the performance and cost of architectures by mapping workloads to the cloud. The current implementation of C-Arch employs LLaMA. We are currently in the process of incorporating a Retrieval Augmented Generation (RAG) pipeline using ChatGPT-4 2(c), aiming to enhance the capabilities of Large Language Models (LLMs) with relevant cloud service information sourced from the internet.

## REFERENCES

[1] Brainboard. [n. d.]. The cloud is your canvas. Accessed Jan. 22, 2024. https://www.brainboard.co/
[2] Cloudcraft. [n. d.]. Visualize your cloud architecture like a Pro. Accessed Jan. 22, 2024. https://www.cloudcraft.co/
[3] Cloudskew. [n. d.]. Online Diagram, Flowchart Maker. Accessed Jan. 22, 2024. https://www.cloudskew.com/
[4] Diagrams.net(Draw.io). [n. d.]. Flowchart Maker Online Diagram Software. Accessed Jan. 22, 2024. https://app.diagrams.net/
[5] Hava.io. [n. d.]. Automated Cloud Diagrams in Minutes. Accessed Jan. 22, 2024. https://www.hava.io/
[6] Holori. [n. d.]. Cloud cost platform with infrastructure visibility. Accessed Jan. 22, 2024. https://holori.com/
[7] Lucidchart. [n. d.]. Diagram your people, processes, and systems. Accessed Jan. 22, 2024. https://www.lucidchart.com/pages/
[8] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

# AIPerf'24: 2nd International Workshop on Artificial Intelligence for Performance Modeling, Prediction, and Control

Emilio Incerto
emilio.incerto@imtlucca.it
IMT School for Advanced Studies Lucca
Lucca, Italy

Marin Litoiu
mlitoiu@yorku.ca
Lassonde School of Engineering, York University,
Canada

Daniele Masti
daniele.masti@imtlucca.it
IMT School for Advanced Studies Lucca
Lucca, Italy


**Figure 1: 2st Workshop on Artificial Intelligence for Performance Modeling, Prediction, and Control**

## CCS CONCEPTS

• **Software and its engineering** → **Software performance**; • **Computing methodologies** → **Control methods**; **Machine learning**.

## KEYWORDS

Software Performance, Control Theory, Artificial Intelligence

**ACM Reference Format:**
Emilio Incerto, Marin Litoiu, and Daniele Masti. 2024. AIPerf'24: 2nd International Workshop on Artificial Intelligence for Performance Modeling, Prediction, and Control. In *Companion of the 15th ACM/SPEC Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3629527.3651433

## 1 WORKSHOP CHAIRS' WELCOME

We are pleased to welcome you to the *2024 ACM Workshop on Artificial Intelligence for Performance Modeling, Prediction, and Control* – AIPerf'24.

In its second edition, AIPerf intends to foster the usage of AI (such as probabilistic methods, machine learning, and deep learning) to control, model, and predict the performance of computer systems. The relevance of these topics reflects current and future trends toward exploiting AI-based approaches to deal with complex, large, and interconnected systems. Despite AI and ML being widely adopted techniques to investigate several mainstream domains, their usage for performance modeling and evaluation is still limited, and their benefit to the performance engineering field remains unclear. AIPerf proposes a meeting venue to promote the dissemination of research works that use or study AI techniques for quantitative analysis of modern ICT systems and to engage academics and practitioners of this field. The workshop focuses on presenting experiences and results of applying AI/ML-based techniques to performance-related problems, as well as sharing performance datasets and benchmarks with the community to facilitate the development of new and more accurate learning procedures.

Remarkably, for this edition, recognizing the strong correlation between the topics, AIPerf is combined with the 1st Workshop on Performance Optimization in the LLM World. We believe that this fusion could offer mutual benefits to the audiences of both workshops, stimulating paper dissemination and fostering fruitful collaborations.

Putting together AIPerf'24 was a team effort. We first thank the authors and the invited speakers for providing the content of the program. We are grateful to the program committee and the senior program committee, who worked very hard to review papers and provide authors' feedback. Finally, we thank the LLM World organizing committee for their help and availability in organizing this joint edition collaboratively and to the ICPE'24 organizers for sponsoring AIPerf in its community. We hope that you will find this program interesting and thought-provoking and that the symposium will provide you with a valuable opportunity to share ideas with other researchers and practitioners from institutions around the world.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom.*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651433

# Benchmarking in the Datacenter (BID): Expanding to the Cloud

Wei-Chen Lin
wl14928@bristol.ac.uk
University of Bristol
Bristol, UK

Jens Domke
jens.domke@riken.jp
RIKEN Center for Computational Science
Kobe, Japan

## ABSTRACT

Welcome to the 2024 5th International Workshop on Benchmarking in the Data Centre: Expanding to the Cloud (BID '24), hosted at Imperial College London as a workshop track of the International Conference on Performance Engineering (ICPE'24).

The past few years have been remarkably exciting for the cloud computing domain. We are witnessing groundbreaking developments in AI architectures, new AI/ML methodologies, and the significant expansion of newer CPU architectures such as AArch64 and RISC-V. These innovations not only redefine the capabilities and efficiency of cloud-based services but also open new avenues of research on how we can attain the best possible performance in the cloud.

BID '24 is dedicated to advancing the field of high-performance computing (HPC) benchmarking, extending its application from traditional academic settings to industry and the cloud. This evolution prompts a reassessment of user education concerning HPC's advantages, optimal selection of computational resources for specific workloads, and the considerations surrounding cost and environmental impact. Our discussions will encompass several key areas: privacy issues in commercial HPC environments, emerging cloud architectures, comprehensive workflows for effective benchmarking, and theoretical approaches to performance analysis. Additionally, this year's workshop will delve into the unique challenges presented by AI/ML workloads in cloud settings.

The success of BID '24 is made possible by the contributions of numerous individuals and organizations. We extend our gratitude to all authors and presenters who have submitted their work for discussion. A special thank you goes to the members of the Technical Committee for their invaluable support and diligent reviews. We also appreciate the hospitality and support from our hosts in London, UK, who have provided an excellent venue for our workshop.

Ultimately, the essence of BID '24 is shaped by its participants. We thank all authors, speakers, and attendees for enriching this workshop with their insights and presence. We hope you find the discussions stimulating, the networking fruitful, and your time in London memorable.

## CCS CONCEPTS

• **Computing methodologies** → **Parallel computing methodologies**; • **Computer systems organization** → **Cloud computing**; **Heterogeneous (hybrid) systems**.

## KEYWORDS

HPC; benchmarking; cloud computing; confidential computing; AI/ML workflow

**ACM Reference Format:**
Wei-Chen Lin and Jens Domke. 2024. Benchmarking in the Datacenter (BID): Expanding to the Cloud . In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '24 ), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3629527.3651434

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE Companion '24 , May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651434

# GraphSys-2024: 2nd Workshop on Serverless, Extreme-Scale, and Sustainable Graph Processing Systems

## Chairs' Welcome

Alexandru Iosup
a.iosup@vu.nl
VU University of Amsterdam
Amsterdam, The Netherlands

Radu Prodan
radu.prodan@aau.at
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria

Ana-Lucia Varbanescu
a.l.varbanescu@utwente.nl
University of Twente
Enschede, The Netherlands

### Welcome!

It is our great pleasure to welcome you to *GraphSys'24*, the *2nd edition of the ACM/SPEC Workshop on Serverless, Extreme-Scale, and Sustainable Graph Processing Systems*. This is a returning workshop, where we continue to facilitate the exchange of ideas and expertise in the broad field of high-performance large-scale graph processing.

### Graphs and GraphSys

The use, interoperability, and analytical exploitation of graph data are essential for modern digital economies. Today, thousands of computational methods (algorithms) and findable, accessible, interoperable, and reusable (FAIR) graph datasets exist. However, current computational capabilities lag when faced with the complex workflows involved in graph processing, the extreme scale of existing graph datasets, and the need to consider sustainability metrics in graph-processing operations. Needs are emerging for graph-processing platforms to provide multilingual information processing and reasoning based on the massive graph representation of extreme data in the form of general graphs, knowledge graphs, and property graphs. Because graph workloads and graph datasets are strongly irregular, and involve one or several big data "Vs" (e.g., volume, velocity, variability, vicissitude), the community needs to reconsider traditional approaches in performance analysis and modeling, system architectures and techniques, serverless and "as a service" operation, real-world and simulation-driven experimentation, etc., and provide new tools and instruments to address emerging challenges in graph processing.

Graphs or linked data are crucial to innovation, competition, and prosperity and establish a strategic investment in technical processing and ecosystem enablers. Graphs are universal abstractions that capture, combine, model, analyze, and process knowledge about real and digital worlds into actionable insights through item representation and interconnectedness. For societally relevant problems, graphs are extreme data that require further technological innovations to meet the needs of the European data economy. Digital graphs help pursue the United Nations Sustainable Development Goals (UN SDG) by enabling better value chains, products, and services for more profitable or green investments in the financial sector and deriving trustworthy insight for creating sustainable communities. All science, engineering, industry, economy, and society-at-large domains can leverage graph data for unique analysis and insight, but only if graph processing becomes easy to use, fast, scalable, and sustainable.

GraphSys is a cross-disciplinary meeting venue focusing on state-of-the-art and the emerging (future) graph processing systems. We invite experts and trainees in the field, across academia, industry, governance, and society, to share experience and expertise leading to a shared body of knowledge, to formulate together a vision for the field, and to engage with the topics to foster new approaches, techniques, and solutions.

### Technical content

In this second edition, the workshop received 10 submissions from which, after peer review, it accepted six full papers for publication. Topics include graph sampling characterization, new models for pipeline/streaming computation on graphs, and using graphs for thermal modeling in HPC centers, telemetry data analysis, and improvements in collition detection for autonomous vehicle driving. We are sure this mix of topics will be reflected in a lively workshop, from presentations to discussions. We are looking forward to it!

We further invited the other four submissions to submit a work-in-progress short papers. Three teams have accepted our invitation, and thus the workshop will feature three short, work-in-progress talks on creating massive knowledge graphs, state-of-the-art in serverless workflow management, and graph-sampling algorithms.

Last, but not least, we are happy to welcome two invited speakers to our workshop. Dr. Johannes Langguth (SIMULA, Norway) will talk about *Graph Algorithms on Emerging Tile-Centric Accelerators*. Dr. Gabor Szarnyas from the LDBC (Linked Data Benchmark Council, The Netherlands) will talk about *Linked Data Benchmark Council: 12 years of fostering competition in the graph processing space*.

### Thanks and Acknowledgments

This year's GraphSys is the result of the collaboration of many people, authors, reviewers, and the organizing committee. Thank you! We further thank the ICPE 2024 Program and Workshop Chairs, for their active support. We could not have done this without you! The GraphSys workshop is technically sponsored by the GraphMassivizer project funded by the Horizon Europe research and innovation program of the European Union for the period 2024-2026,

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE Companion '24 , May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651435

which studies and aims to develop a high-performance, scalable, gender-neutral, secure, and sustainable platform for massive graph processing.

## Concluding remarks

We aim to continue to align GraphSys with the Standard Performance Evaluation Corporation (SPEC)'s Research Groups (RG), and, in particular, the RG Cloud Group that is taking a broad approach, relevant for both academia and industry, to cloud benchmarking, quantitative evaluation, and experimental analysis.

We hope GraphSys will continue its growth towards a focused yearly series, aiming to develop the topic of large scale high performance, sustainable graph processing, and around it a community of knowledge and practice.

# Linked Data Benchmark Council: 12 years of fostering competition in the graph processing space

Gábor Szárnyas
gabor.szarnyas@ldbcouncil.org
Linked Data Benchmark Council
The Netherlands

## ABSTRACT

The Linked Data Benchmark Council (LDBC) was originally created in 2012 as part of a European Union-funded project of the same name. Its original goal was to design standard benchmarks for graph processing and to facilitate competition among vendors to drive innovation in the field. 12 years later, the LDBC organization has 20+ member organizations (including database, hardware, and cloud vendors) and has five standard benchmark workloads, with frequent audit requests from vendors. Moreover, LDBC extended its scope behind benchmarking to cover graph schemas and graph query languages, and has a liaison arrangement with the ISO SQL/GQL standards committee. In this talk, I will reflect on the LDBC's organizational history, goals, and main technical achievements.

**ACM Reference Format:**
Gábor Szárnyas. 2024. Linked Data Benchmark Council: 12 years of fostering competition in the graph processing space. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3629527.3652889

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652889

# GraphMa: Towards new Models for Pipeline-Oriented Computation on Graphs

Daniel Thilo Schroeder
SINTEF
Norway
daniel.t.schroeder@sintef.no

Tobias Herb
Germany
tobias.herb@gmx.de

Brian Elvesæter
SINTEF
Norway
brian.elvesater@sintef.no

Dumitru Roman
SINTEF
Norway
dumitru.roman@sintef.no

## ABSTRACT

This paper presents GraphMa, a framework aimed at enhancing pipeline-oriented computation for graph processing. GraphMa integrates the principles of pipeline computation with graph processing methodologies to provide a structured approach for analyzing and processing graph data. The framework defines a series of computational abstractions, including computation as type, higher-order traversal, and directed data-transfer, which collectively facilitate the decomposition of graph operations into modular functions. These functions can be composed into pipelines, supporting the systematic development of graph algorithms. For this paper, our focus lies in particular on the capability to implement the well-established computational models for graph processing within the proposed framework. In addition, the paper discusses the design of GraphMa, its computational models, and the implementation details that illustrate the framework's application to graph processing tasks.

## CCS CONCEPTS

• **Computer systems organization → Pipeline computing**.

## KEYWORDS

Graph processing, Pipeline-oriented computation, Graph data, Graph operations

**ACM Reference Format:**
Daniel Thilo Schroeder, Tobias Herb, Brian Elvesæter, and Dumitru Roman. 2024. GraphMa: Towards new Models for Pipeline-Oriented Computation on Graphs. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3652894

Tobias Herb and Daniel Thilo Schroeder contributed equally.

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652894

## 1 INTRODUCTION

Graphs, with their intricate structures and complex relationships, are an integral part of the world around us, playing a key role in various domains ranging from social networks to biological systems. Historically, graph processing has been fundamental in applications such as network analysis, e.g. shortest path computation [12, 22], for mapping services [5], analysis of social networks [13, 21] or even feature extraction [17]. These traditional uses have paved the way for more advanced applications.

In recent times, we have witnessed the emergence of graph processing in areas like recommendation systems [3, 4, 7, 20], fraud detection [11], and complex interaction mapping in bioinformatics. Moreover, the increasing availability and collection of large datasets have significantly influenced the size and complexity of graphs [16]. These large-scale graphs present unique challenges and opportunities for processing and analysis.

In response to these challenges, distributed graph processing has gained prominence [2]. This shift from traditional, localized graph processing to distributed methods addresses the need for scalability and efficiency in handling vast and more and more complex graph structures. Frameworks like Apache Giraph [10], Google's Pregel [9] or the Apache TinkerPop [14] project have been instrumental in this transition. They offer powerful, general-purpose solutions for distributed graph processing, enabling easier implementation of algorithms tailored to specific graph-related problems.

Building upon the foundational aspects of graph processing, the application of concepts like immutability or modularity, well-known in functional programming, offer a promising approach for constructing graph processing pipelines. The congruence between these concepts and graph processing is rooted in several advantages, which are particularly beneficial for handling the challenges posed by graph data.

In this context, modularity is the decomposition of complex problems into smaller, reusable functions. This approach mirrors the inductive nature of many graph algorithms and processing workflows, where operations are independent yet interrelated. In graph processing, this translates to the ability to encapsulate operations like traversal, filtering, and transformation into discrete functions, which can then be composed to address more complex graph-related problems. Such modularity not only fosters code reusability but also simplifies the process of constructing and maintaining graph

algorithms and even beyond, the composition of a multitude of graph algorithms.

Graph processing, especially when dealing with large datasets, can greatly benefit from parallel and concurrent execution. Reducing or eliminating shared mutable states, simplifies the management and reasoning of parallelism and concurrency. This is crucial for optimizing the performance of graph algorithms, allowing them to handle the computational demands of large-scale graph analysis. The immutability and stateless nature embedded in the concepts of functional programming make it inherently suited for these tasks.

In this paper, we outline the fundamental principles that underlay our concept for pipeline-oriented computation in general. Building on this we propose and discuss GraphMa, a collection of ideas and preliminary implementations extending the ideas around general pipeline-oriented computation towards graph processing. For this paper our focus lies in particular on the capability to implement the well-established computational models for graph processing (see Section 2) within the proposed framework (see Section 3).

## 2 BACKGROUND: COMPUTATIONAL MODELS FOR GRAPH PROCESSING

The landscape of graph processing is rich and varied, encompassing a range of computational units and models each designed to optimize different aspects of graph analysis. At the heart of these models is the goal to efficiently process and analyze data structured in graphs. Based on [2] we would like to exercise an overview of the primary computational models that have shaped modern graph processing, highlighting their foundations, operations, and the challenges they address.

- *Vertex-Centric (TLAV) Model:* Pioneered by Google's Pregel [9] and further extended by Apache Giraph [1], the Vertex-Centric model places the vertex at the center of computation. In this model, each vertex independently executes the same function, processing incoming information, potentially updating its state, and then communicating with other vertices through its edges. This approach allows for high levels of parallelism as each vertex operates in isolation, yet collaboratively contributes to the graph's overall computation.
- *Superstep Paradigm:* This execution model, integral to Vertex-Centric processing, organizes computation into a series of global steps known as supersteps. During a superstep, vertices concurrently execute a specified function, after which they engage in communication by sending messages to vertices that will be active in the subsequent superstep. This synchronized execution and communication phase structure not only facilitates easier reasoning about the computational process and thus simplifies the programming of distributed graph algorithms. The paradigm is briefly described in [2]. We also recommend read [19] and [9].
- *Scatter-Gather Model:* This model splits the process of message handling into two distinct phases: scattering, where vertices send out messages, and gathering, where messages are collected and state updates are aggregated. By clearly distinguishing between these phases, the Scatter-Gather model [18] provides a structured approach to handling vertex communication, facilitating more organized data processing flows.
- *Gather-Apply-Scatter Model:* Introduced by PowerGraph [6] adresses the challenge of computational load imbalance, especially in graphs with power-law distributions, the Gather-Apply-Scatter model decomposes vertex operations into three phases: gather information from neighboring vertices, apply a function to update the vertex's state, and scatter results to influence neighboring vertices in the next cycle.
- *Edge-Centric Model:* Offering a different perspective, the Edge-Centric model [15] focuses computation on graph edges rather than vertices. This model, exemplified by X-Stream and Chaos, is particularly effective in scenarios where edge-based computations are predominant. This model optimizes the use of secondary storage and network communication, making it suitable for processing very large graphs that do not fit into memory.
- *Sub-graph-Centric Model:* By concentrating on sub-graphs, either partition-centric within a physical partition or neighbourhood-centric allowing for shared state updates, this model [8] aims to reduce communication overheads. This approach is especially beneficial in distributed environments where minimizing inter-node communication can significantly enhance performance.
- *MEGA Model:* Specifically designed for machine learning applications on graphs, the MEGA model introduced by Tux2 [23] focuses on edge-level computations with functions such as Exchange, Apply, and Global Sync. These functions facilitate detailed manipulation of graph structure and values, supporting sophisticated machine learning algorithms on graph data.

## 3 A CONCEPTUAL FRAMEWORK TO PIPELINE-ORIENTED COMPUTATION

This section presents a comprehensive overview of our novel computation model designed for pipeline-oriented data processing in general. It is only in Section 4 when we discuss how to apply this framework as the basis to implement computational models for graph processing. Our proposed model integrates functional programming paradigms with object-oriented design principles to create a versatile framework capable of addressing complex data processing requirements. It is structured around a series of interconnected layers, each contributing to a cohesive and flexible architecture that facilitates the development of data processing pipelines. These layers include *Computation as Type*, *Higher-order Traversal Abstraction*, *Directed Value-Transfer Protocol*, *Operator Model*, and finally the Pipeline Abstraction. We begin to introduce the first Layer *Computation as Type*.

### 3.1 Computation as Type

The foundation of our model is the concept of Computation as Type, which posits computation units as first-class entities encapsulated by a `Compute` interface. This interface is defined as a function accepting a value of type `T` and performing operations, potentially

with side effects. This foundational abstraction underpins the architecture for creating pipeline stages, enabling data processing that is both type-safe and modular.

## 3.2 Higher-order Traversal Abstraction

The second layer, Higher-order Traversal Abstraction is the component designed to oversee data access and processing in an abstract manner. It outlines the methods for navigating through different data sources, thereby enabling versatile and effective data manipulation. In this section, we explore the fundamental elements of this abstraction, detailing their organizational framework and how they interact.

*3.2.1 Structural Composition and Behavioral Interaction.*

*Higher-order Traversal Primitives:* At the core of our Traversal Abstraction are the higher-order traversal primitives, which are instrumental in defining the manner in which data is accessed and iterated over. These primitives are characterized by their ability to abstract away the specifics of data sources and access mechanisms, providing a unified interface for data traversal. Key characteristics include:

- *Sequential Access*: An abstraction layer that decouples the traversal mechanism from the data source's physical representation while allowing for sequential access to data, enabling iteration over data sources like containers, IO channels, or generator functions.
- *Computation Integration*: An important feature is the integration with first-order computation units, allowing traversed values to be processed in a seamless and flexible manner. This is achieved through second-order functions that pass each traversed value to a specified computation unit (`Compute<T>`).
- *Traversal Strategies*:
  - *Single-step Traversal (*`TryNext`*)*: Processes data one item at a time, affording precise control over the iteration and enabling fine-grained data manipulation.
  - *Bulk Traversal (*`ForNext`*)*: Optimizes data processing by handling batches of data, streamlining the traversal process and improving efficiency.
  - *Continuation-controlled Bulk Traversal (*`WhileNext`*)*: Introduces a continuation-passing style for bulk processing, offering dynamic control over the traversal logic based on runtime conditions. This strategy is particularly notable for its use of the `Continuation` interface, which provides a mechanism for halting or altering the course of computation in response to specific criteria.

*Traverser Abstraction in a nutshell:*

- *Computation Carrier*: The `Traverser` emerges as the central figure in this abstraction, acting as the carrier for the computation across data sets. It encapsulates the higher-order traversal primitives, serving as the execution context for data processing operations.
- *Unified Control Flow Patterns*: By housing different traversal strategies within a coherent framework, the `Traverser` harmonizes flexibility with control. It offers a spectrum of control flow patterns, from granular, step-by-step data processing to more coarse-grained, bulk handling techniques.
- *Seamless Interaction and Modularity*: The delineation of traversal strategies into distinct components not only clarifies the traversal abstraction but also enhances the system's modularity. This separation allows for the extension and customization of traversal behaviors to accommodate specific processing requirements, fostering reusability and adaptability.
- *Foundation for Advanced Data Processing*: The integration of traversal primitives within a unified `Traverser` environment provides a robust foundation for implementing sophisticated data processing strategies. This design carefully balances the need for complex control flow mechanisms with the desire for a clear, modular architectural structure.

By abstracting the intricacies of data traversal and offering a suite of customizable traversal strategies, the Traverser layer stands as a cornerstone of the proposed computation model. It exemplifies the framework's capacity to facilitate advanced data manipulation techniques.

## 3.3 Directed Data-Transfer Protocol

We introduce a refined conceptual model for pipeline-oriented computation, anchored by the Directed Value-Transfer Protocol. This model emphasizes the seamless management and transfer of data across computational stages, aligning with the principles of functional programming and type-centric design philosophies. Our model is distinguished by its lifecycle-aware architecture and the explicit delineation of data producer and consumer roles, facilitating a structured yet flexible approach to constructing computation pipelines.

*3.3.1 Architectural Foundations.* At the heart of our model lies the `Compute<T>` interface, a fundamental abstraction representing a unit of computation. It encapsulates the notion that computations are first-class entities, capable of accepting input and executing operations in a type-safe manner. Building upon this, our design introduces a hierarchical structure aimed at enhancing data transfer efficiency and lifecycle management:

- *Lifecycle Integration*: The `Transfer.Lifecycle` interface introduces a dual-phase lifecycle management protocol with `open()` and `close()` methods. This protocol ensures the acquisition and release of resources are handled gracefully, enhancing the robustness of the computation chain.
- *Role-Specific Abstractions*: Our model defines two important interfaces, `Transfer.Port<T>` and `Transfer.Pipe<T>`, to represent the roles of data producers and consumers, respectively. This distinction not only clarifies the data flow directionality, but also enriches the model with the capability to handle complex data processing scenarios.

*3.3.2 Operational Dynamics.* The Directed Value-Transfer Protocol underpins an interaction framework:

- *Managed Data Flow*: The explicit lifecycle management embedded within the data transfer interfaces ensures that each stage of the computation pipeline is initialized and terminated appropriately, promoting efficient resource utilization.

- *Directed Transfer Mechanism*: By segregating data transfer roles into `Port` and `Pipe`, our model ensures a clear and efficient directionality of data flow. This segregation allows for the optimization of data transmission mechanisms, catering to both synchronous and asynchronous processing needs.
- *Flexible Computation Chains*: The extension of `Pipe<T>` to potentially act as both consumer and producer underscores the model's versatility. It supports the construction of intricate multi-stage processing pipelines, enabling data to be transformed progressively through successive computational units.

*3.3.3 Computational Chain Composition.* A key feature of our model is the `LazyChain<I,O>` interface, which symbolizes the essence of pipeline-oriented computation through the contravariant composition of computation units. This interface facilitates the dynamic assembly of computational stages, allowing for the efficient transformation and transfer of data across the pipeline:

- *Enhanced Modularity*: The `LazyChain` interface exemplifies the model's commitment to modular and reusable design principles. It allows for the flexible chaining of computational units, ensuring that complex data processing tasks can be decomposed into manageable, composable segments.

The Directed Value-Transfer Protocol, as conceptualized in our pipeline-oriented computation model, represents a sophisticated framework for data processing. It marries the principles of lifecycle management, type safety, and functional programming to offer a robust and flexible solution for constructing complex computational pipelines. Through this model, we aim to provide a scalable and efficient framework for addressing the diverse challenges of modern data processing tasks, reaffirming the potential of functional patterns in the realm of object-oriented programming languages like Java.

## 3.4 Operator Model

The Operator Model represents the quintessential fifth layer within our innovative pipeline-oriented computational framework, meticulously crafted to underpin the construction and orchestration of data processing pipelines. This model introduces a sophisticated suite of computational constructs, pivotal for the lifecycle management of operators and the nuanced handling of their states, thereby facilitating a broad spectrum of data processing operations. Herein, we integrate and refine the abstract conceptualization of the Operator Model, emphasizing its core constructs, their structural interplay, and the pivotal role of terminal operators in concluding data processing tasks.

*3.4.1 Core Constructs and Structural Composition.*

*Operator Protocol.* The `Operator<T>` abstraction stands as the cornerstone of the Operator Model, extending Transfer.Lifecycle to underscore its essential role in managing the lifecycle and state of operations within pipelines. This interface is instrumental for:

- *State Management*: It allows for the encapsulation of stateful computations, enabling operators to maintain and manipulate local state through the `localState()` method.

*Transducer.* The `Transducer<I, O>` abstraction, serving as the backbone for intermediate operators, embodies the transformational logic necessary for processing and relaying data through various stages of the pipeline. Its design is focused on:

- *Transformation and Lazy Computation*: Facilitating the lazy transformation of data, thus acting as a critical bridge in the data flow across the pipeline.

*Materializer.* The `Materializer<T>` abstraction plays a crucial role in state materialization, especially in managing the transition of data states within pipelines through chunked buffers, enhancing the efficiency and organization of data processing workflows.

*Terminal Operators.* Terminal operators, categorized into Complete Terminal Operators and Partial Terminal Operators, mark the culmination of the pipeline's data processing journey. They are distinguished by their evaluation strategies:

- *Complete Terminal Operators* process the entirety of input data, embodying exhaustive data analysis or transformation.
- *Partial Terminal Operators* facilitate early termination of processing based on specific conditions, optimizing performance through lazy evaluation and early termination strategies.

*3.4.2 Interaction Dynamics and Evaluation Strategies.*

- *Lifecycle and State Management*: The Operator Model ensures meticulous lifecycle management across all operator types, harmonizing state management and data transformation processes. This integration is vital for the seamless flow and transformation of data across the pipeline.
- *Flexible Terminal Evaluation*: The differentiation in terminal operator strategies enhances the model's adaptability, allowing for both exhaustive data processing and efficient, condition-based evaluations. This flexibility ensures optimal performance and resource utilization, catering to a wide range of computational requirements.

The Operator Model emerges as a comprehensive and modular framework for pipeline construction, characterized by its advanced management of operator lifecycle, state, and terminal evaluation strategies. Through its well-structured abstractions—from transducers and materializers to the nuanced categorization of terminal operators—it lays a versatile and extensible foundation for domain-specific data processing operations. This model not only encapsulates the core principles of pipeline-oriented computation but also fosters adaptability and efficiency, ensuring its applicability across diverse data processing scenarios.

## 3.5 Pipeline Abstraction

The `Pipeline` abstraction is a pivotal construct within the novel pipeline-oriented computational model, representing the overarching framework that orchestrates the structured and stateful processing of data. This abstraction serves as a high-level blueprint for defining data processing flows, encapsulating the complexities of data transformation and transmission. The design principles underlying the `Pipeline` model prioritize modularity, flexibility, and clarity in constructing computational logic, thereby offering a robust platform for implementing sophisticated data processing mechanisms.

*3.5.1 Overview of Pipeline Components.*

- *Stage*: A `Stage`within the pipeline signifies a discrete processing unit, tasked with receiving input values, applying a specified operation, and producing output for the subsequent stage. It embodies the core functional element of the pipeline, enabling the definition and execution of transformation operations in a type-safe manner.
- *State*: The `State` component encapsulates stateful logic for data materialization, transforming or accumulating data as it traverses the pipeline. This aspect of the pipeline architecture facilitates the implementation of complex data processing semantics, allowing for the dynamic evolution of computation based on the flow of data.
- *Sink*: Serving as the terminal point of the pipeline, the `Sink` is responsible for consuming all processed data to produce a final outcome or effectuate a side operation. It marks the culmination of the pipeline's computational process, transitioning the abstract pipeline description into an actionable computation through the evaluation operator.

*3.5.2 Structural Composition.* The pipeline is conceptually structured as a series of computation steps, organized as `LazyChain` instances and interconnected via `Transfer.Pipe` objects. These steps converge at an `Operator.Terminal`, where the computed data is either transformed into a result or utilized to perform a side effect. The recursive type parameterization of the `Pipeline` interface ensures type safety across the processing stages, facilitating the seamless chaining of operations.

*3.5.3 Interaction and Evaluation Strategies.* The pipeline model embraces the principle of "laziness," deferring computations until absolutely necessary. This design choice enables the efficient assembly of an execution plan that outlines the data transformation process, from the source through to the sink. The plan encapsulates the requisite parameters for executing computations at each stage, culminating in a pipeline sink where the evaluation operator resides.

The evaluation operator, embodied by the `Evaluator`, allows for the implementation of various evaluation strategies:

- *Eager Evaluation*: Immediate execution of computations upon their invocation.
- *Lazy Evaluation*: Deferral of computations until required, encapsulated within a 'thunk' to capture the deferred computation.
- *Memoized Evaluation*: Computations are performed upon first access, with results cached for future reference.

This flexible evaluation framework, referred to as the 'Flow-Machine', grants granular control over the computation flow, enhancing resource utilization and potentially improving execution speed and memory efficiency depending on the use case scenario.

The `Pipeline` abstraction forms the crux of a sophisticated computational model designed to facilitate the structured and stateful processing of data. Through its modular composition, the pipeline model enables the construction of complex data processing flows with ease, offering a comprehensive framework for the implementation of diverse computational logic. This abstraction not only simplifies the development of data processing applications but also enriches the computational model with a flexible and powerful mechanism for data transformation and evaluation.

# 4 IMPLEMENTATION OF COMPUTATIONAL MODELS FOR GRAPH PROCESSING IN GRAPHMA

In this section we propose approaches on how to embed graph computation models into the higher-order pipeline model.

## 4.1 Vertex-Centric Embedding

Embedding the *Vertex-Centric Computation Model* (TLAV) into the higher-order pipeline model could leverage the strengths of both models to efficiently process graph-based data.

Below we give a concise overview of how this integration is currently structured and operates.

*4.1.1 Structural Composition.*

(1) **Compute<T>for Vertex Operations**: Vertices are encapsulated as `Compute<Vertex>` instances, where each vertex acts as an independent computational unit with its own state. This aligns with the Computation as Type principle, allowing vertices to process data and messages in a type-safe manner.
(2) **Traversal as Message Passing**: The Higher-order Traversal Abstraction is adapted to facilitate message passing between vertices. Each vertex employs traversal primitives to send and receive messages, abstracting the communication mechanism and ensuring flexibility in message dissemination strategies.
(3) **Directed Data-Transfer for Supersteps**: The Directed Data-Transfer Protocol orchestrates the execution of supersteps. A `Transfer.Pipe<Message>` interface manages the asynchronous delivery of messages between supersteps, ensuring that messages sent in one superstep are correctly queued for processing in the next.
(4) **Operator Model for Vertex Execution Logic**: The Operator Model is extended to define vertex execution logic within supersteps. `Operator<Vertex>` interfaces manage state transitions and message processing, supporting the iterative nature of vertex-centric computations.
(5) **Pipeline Abstraction for Graph Processing Flows**: The entire graph processing logic is encapsulated within a `Pipeline<Graph>` abstraction, orchestrating the flow of computation across supersteps. This pipeline integrates stages for message passing, vertex state updates, and global convergence checks.

*4.1.2 Behavioral Interaction.*

(1) **Iterative Pipeline Stages**: Each superstep is represented as a stage in the pipeline, with vertices operating in parallel to process incoming messages and update their states. The pipeline dynamically adapts to the iterative nature of the vertex-centric model, allowing for repeated execution of stages until a global stopping condition is met.
(2) **Dynamic Message Routing**: The pipeline utilizes the Higher-order Traversal Abstraction to dynamically route messages

between vertices. This enables efficient scatter-gather operations, optimizing the distribution and collection of messages across the graph.

(3) **Stateful Computation and Lifecycle Management**: The Directed Data-Transfer Protocol and Operator Model jointly manage the lifecycle of vertex computations. They ensure that vertex states are correctly initialized, updated, and finalized across supersteps, maintaining consistency and robustness in the computation.

(4) **Flexible Evaluation Strategies**: The Pipeline Abstraction supports flexible evaluation strategies for vertex-centric computations, allowing for both eager and lazy execution of supersteps. This flexibility aids in optimizing performance based on the specific characteristics of the graph and the computational workload.

(5) **Adaptation to Vertex-Centric Variations**: The pipeline model's modular design allows for easy adaptation to different vertex-centric variations (bulk synchronous parallel, asynchronous, edge-centric, mixed-mode). Specific components of the pipeline (e.g., message routing, state management) can be customized to reflect the desired computational semantics and performance characteristics.

*4.1.3 Summary.* By integrating the Vertex-Centric Computation Model within the higher-order pipeline model, this approach provides a structured yet flexible platform for graph processing.

It combines the modularity, type safety, and functional programming strengths of the pipeline-oriented model with the intuitive, scalable, and vertex-focused computation of the vertex-centric model.

This integration not only enhances the expressiveness and efficiency of graph processing tasks but also leverages the parallel execution capabilities inherent in distributed computing environments.

## 4.2 Edge-Centric Embedding

Embedding the *Edge-Centric Computation Model* (TLAE) into the higher-order pipeline model involves focusing on the relationships and interactions between vertices, with edges acting as the primary conduits of computation and communication.

Below we give a concise overview of how such an integration could be structured and operate.

*4.2.1 Structural Composition.*

(1) **Compute<Edge>for Edge Operations**: Edges are represented as `Compute<Edge>` instances, where each edge acts as an independent computational unit capable of accessing and modifying the data of its connected vertices as well as its own properties. This encapsulation aligns with the Computation as Type principle, facilitating type-safe edge operations.

(2) **Directed Data-Transfer for Edge-Vertex Communication**: A `Transfer.Pipe<Message>` interface is utilized for edge-to-vertex message passing, managing the asynchronous exchange of messages. This supports direct communication between edges and vertices, allowing edges to send messages that influence vertex state and behavior.

(3) **Operator Model for Edge Execution Logic**: The Operator Model is adapted to define the logic of edge computations within supersteps. `Operator<Edge>` interfaces handle the processing tasks of edges, including state transitions based on both edge properties and received vertex messages.

(4) **Pipeline Abstraction for Edge-Centric Flows**: The graph processing logic, focusing on edge interactions, is encapsulated within a `Pipeline<Edge>` abstraction. This pipeline manages the stages of edge computation, message passing, and the integration of edge-induced vertex updates.

*4.2.2 Behavioral Interaction.*

(1) **Iterative Pipeline Stages for Edges**: Supersteps are represented as stages in the pipeline, with edges performing computation and message passing in parallel. This approach facilitates the autonomous operation of edges, allowing for the iterative processing of edge and vertex interactions across supersteps.

(2) **Dynamic Edge-to-Vertex Message Routing**: Utilizing the Directed Data-Transfer Protocol, the pipeline dynamically routes messages from edges to their connected vertices. This mechanism is crucial for enabling edges to influence vertex state and initiate vertex-level computations based on edge-centric logic.

(3) **Stateful Edge Computation and Lifecycle Management**: The integration of the Operator Model with the Directed Data-Transfer Protocol ensures that edge states are correctly managed throughout the computation lifecycle. This includes initialization, state updates based on incoming messages, and finalization, maintaining robustness and consistency in edge-centric computations.

(4) **Flexible Evaluation Strategies for Edge-Centric Operations**: The Pipeline Abstraction supports both eager and lazy execution strategies for edge-centric computations. This flexibility allows for performance optimization based on the graph's characteristics and the computational workload, enhancing the efficiency of edge-centric processing.

(5) **Adaptation to Edge-Centric Variations**: The modular design of the model facilitates easy adaptation to different edge-centric variations (pure edge-centric, vertex-augmented, hybrid). Specific pipeline components (e.g., message routing, edge computation logic) can be tailored to reflect the computational semantics and performance characteristics desired for each variation.

*4.2.3 Summary.* By embedding the Edge-Centric Computation Model within the higher-order pipeline model, this approach offers a structured platform for focusing on edge-based interactions and data flows in graph processing.

It leverages the modularity, type safety, and functional programming benefits of the pipeline-oriented model, along with the direct communication and computation capabilities inherent in edge-centric approaches.

This integration not only provides a powerful tool for addressing problems where edge relationships are paramount but also enhances the flexibility and performance of graph processing tasks in distributed computing environments.

## 4.3 Sub-Graph-Centric Embedding

Embedding the Sub-Graph-Centric Computation Model (TLAG) into the pipeline-oriented computation model focuses on leveraging cohesive groups within the graph structure, enhancing computational efficiency and expressiveness. This integration aims to exploit the locality and reduce communication needs by treating subgraphs as primary computational units.

Below we explain how such an integration could be structured and operate.

*4.3.1 Structural Composition.*

(1) **Compute<SubGraph>for Subgraph Operations**: Subgraphs are encapsulated as `Compute<SubGraph>` instances, treating each subgraph as an independent computational unit. This approach aligns with the Computation as Type principle, facilitating type-safe operations on subgraph data, including both its internal structure and interconnections.
(2) **Directed Data-Transfer for Subgraph Communication**: Utilizing the `Transfer.Pipe<Message>` interface, the model manages asynchronous message passing between subgraphs. This enables subgraphs to communicate, exchanging information and updates in a manner that respects the locality of data and computations.
(3) **Operator Model for Subgraph Execution Logic**: The Operator Model adapts to define subgraph-level computations. `Operator<SubGraph>` interfaces are responsible for executing computations that consider the subgraph's entire structural and relational context, supporting iterative execution patterns for dynamic state updates.
(4) **Pipeline Abstraction for Subgraph-Centric Processing**: The graph processing logic, centered around subgraph computations, is encapsulated within a `Pipeline<SubGraph>` abstraction. This pipeline coordinates the stages of subgraph computation, communication, and integration of updates across the larger graph structure.

*4.3.2 Behavioral Interaction.*

(1) **Iterative Pipeline Stages for Subgraphs**: Each pipeline stage corresponds to a superstep in the subgraph-centric computation, allowing subgraphs to process information and interact in parallel. This iterative approach facilitates the dynamic exchange of information and updates across subgraphs, maintaining the model's emphasis on locality and reduced communication overhead.
(2) **Dynamic Subgraph-to-Subgraph Communication**: The pipeline uses the Directed Data-Transfer Protocol to enable efficient and localized message passing between subgraphs. This setup is crucial for maintaining data locality and reducing communication overhead, particularly for algorithms that benefit from intensive local interactions.
(3) **Stateful Subgraph Computation and Lifecycle Management**: Integrating the Operator Model with the Directed Data-Transfer Protocol ensures that subgraph computations are managed effectively throughout their lifecycle. This includes initialization, iterative processing based on structural and relational context, and finalization, ensuring consistency and robustness in subgraph-centric computations.
(4) **Flexible Evaluation Strategies for Subgraph-Centric Operations**: The Pipeline Abstraction supports various evaluation strategies, including eager and lazy execution, tailored to the computational needs of subgraph-centric processing. This flexibility allows for optimization of performance based on the graph's structure and the computational workload, enhancing the efficiency of processing within and across subgraphs.
(5) **Adaptation to Subgraph-Centric Variations**: The model's modular design facilitates easy adaptation to different subgraph-centric variations (TLAG, graph-centric, neighbourhood-centric, hybrid). Specific pipeline components can be customized to reflect the computational semantics and performance characteristics desired for each variation, ensuring that the approach is tailored to the specific requirements of the problem at hand.

*4.3.3 Summary.* By embedding the Sub-Graph-Centric Computation Model within the higher-order pipeline model, this approach offers a structured yet flexible platform for focusing on computations within cohesive subgraph units.

It leverages the strengths of the pipeline-oriented model – modularity, type safety, and functional programming benefits – along with the enhanced locality, reduced communication needs, and expressiveness of the subgraph-centric approach.

This integration not only provides a powerful mechanism for addressing graph processing challenges that benefit from subgraph-level focus but also enriches the computational model with advanced capabilities for handling complex patterns and relationships in large-scale graph data.

# 5 CONCLUSION

In this paper, we introduced GraphMa, a collection of ideas and preliminary implementations extending the ideas around general pipeline-oriented computation towards graph processing. We argued that GraphMa's architecture, which merges pipeline computation principles with graph processing techniques, provides a structured method for constructing and executing graph algorithms. Through the introduction of computational abstractions such as computation as type, higher-order traversal abstraction, and directed data-transfer protocol, GraphMa enables the decomposition of complex graph operations into modular, composable functions.

Furthermore, we have qualitatively explored the potential integration of well-established computational models for graph processing within the GraphMa framework. This exploration has highlighted the framework's inherent flexibility and the theoretical effectiveness of such an integration. By detailing how these computational models could align with GraphMa's pipeline-oriented architecture, we have shed light on the framework's potential to facilitate and enhance the development and execution of graph processing tasks.

Looking ahead, we anticipate that GraphMa will serve as a valuable tool for researchers and practitioners in the field of graph processing, offering a scalable and modular approach to algorithm development. Future work will involve extending GraphMa's capabilities, exploring its application to a broader range of graph processing scenarios, and evaluating its performance in comparison

to existing frameworks. Our goal is to continue refining GraphMa, ensuring that it remains a robust and adaptable framework capable of addressing the evolving challenges in graph processing.

## ACKNOWLEDGMENT

This work has been funded by the Graph-Massivizer project, which receives funding from the Horizon Europe research and innovation program of the European Union under grant agreement No 101093202.

## REFERENCES

[1] Avery Ching. 2013. Scaling apache giraph to a trillion edges.
[2] Miguel E Coimbra, Alexandre P Francisco, and Luís Veiga. 2021. An analysis of the graph processing landscape. *journal of Big Data* 8, 1 (2021), 1–41.
[3] Xiaohui Cui, Xiaolong Qu, Dongmei Li, Yu Yang, Yuxun Li, and Xiaoping Zhang. 2023. MKGCN: Multi-Modal Knowledge Graph Convolutional Network for Music Recommender Systems. *Electronics* 12, 12 (2023), 2688.
[4] Leyan Deng, Defu Lian, Chenwang Wu, and Enhong Chen. 2022. Graph Convolution Network based Recommender Systems: Learning Guarantee and Item Mixture Powered Strategy. *Advances in Neural Information Processing Systems* 35 (2022), 3900–3912.
[5] Lau Nguyen Dinh. 2024. The MapReduce based approach to improve the all-pair shortest path computation. *International Journal of Advanced Science and Computer Applications* 3, 1 (2024), 55–64.
[6] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, Chandu Thekkath and Amin Vahdat (Eds.). USENIX Association, Hollywood, CA, USA, 17–30. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/gonzalez
[7] Elvin Isufi, Matteo Pocchiari, and Alan Hanjalic. 2021. Accuracy-diversity trade-off in recommender systems via graph convolutions. *Information Processing & Management* 58, 2 (2021), 102459.
[8] Vasiliki Kalavri, Vladimir Vlassov, and Seif Haridi. 2017. High-level programming abstractions for distributed graph processing. *IEEE Transactions on Knowledge and Data Engineering* 30, 2 (2017), 305–324.
[9] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, Ahmed K. Elmagarmid and Divyakant Agrawal (Eds.). ACM, Indianapolis, IN, USA, 135–146. https://doi.org/10.1145/1807167.1807184
[10] Claudio Martella, Roman Shaposhnik, Dionysios Logothetis, and Steve Harenberg. 2015. *Practical Graph Analytics with Apache Giraph* (1 ed.). Vol. 1. Apress, Berkeley, CA, Berkeley, CA. XIX, 315 pages. https://doi.org/10.1007/978-1-4842-1251-6
[11] Tahereh Pourhabibi, Kok-Leong Ong, Booi H Kam, and Yee Ling Boo. 2020. Fraud detection: A systematic literature review of graph-based anomaly detection approaches. *Decision Support Systems* 133 (2020), 113303.
[12] Yu-Xuan Qiu, Dong Wen, Lu Qin, Wentao Li, Ronghua Li, and Ying Zhang. 2022. Efficient Shortest Path Counting on Large Road Networks. *Proc. VLDB Endow.* 15, 10 (2022), 2098–2110. https://doi.org/10.14778/3547305.3547315
[13] Louise Quick, Paul Wilkinson, and David Hardcastle. 2012. Using Pregel-like Large Scale Graph Processing Frameworks for Social Network Analysis. In *International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2012, Istanbul, Turkey, 26-29 August 2012*. IEEE Computer Society, Istanbul, Turkey, 457–463. https://doi.org/10.1109/ASONAM.2012.254
[14] Marko A. Rodriguez. 2015. The Gremlin graph traversal machine and language (invited talk). In *Proceedings of the 15th Symposium on Database Programming Languages, Pittsburgh, PA, USA, October 25-30, 2015*, James Cheney and Thomas Neumann (Eds.). ACM, Pittsburgh, PA, USA, 1–10. https://doi.org/10.1145/2815072.2815073
[15] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. 2013. X-Stream: edge-centric graph processing using streaming partitions. In *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*, Michael Kaminsky and Mike Dahlin (Eds.). ACM, Farmington, PA, USA, 472–488. https://doi.org/10.1145/2517349.2522740
[16] Daniel Thilo Schroeder, Johannes Langguth, Luk Burchard, Konstantin Pogorelov, and Pedro G Lind. 2022. The connectivity network underlying the German's Twittersphere: a testbed for investigating information spreading phenomena. *Scientific reports* 12, 1 (2022), 4085.
[17] Daniel Thilo Schroeder, Kevin Styp-Rekowski, Florian Schmidt, Alexander Acker, and Odej Kao. 2019. Graph-based Feature Selection Filter Utilizing Maximal Cliques. In *Sixth International Conference on Social Networks Analysis, Management and Security, SNAMS 2019, Granada, Spain, October 22-25, 2019*, Mohammad A. Alsmirat and Yaser Jararweh (Eds.). IEEE, Granada, Spain, 297–302. https://doi.org/10.1109/SNAMS.2019.8931841
[18] Philip Stutz, Abraham Bernstein, and William W. Cohen. 2010. Signal/Collect: Graph Algorithms for the (Semantic) Web. In *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I (Lecture Notes in Computer Science, Vol. 6496)*, Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm (Eds.). Springer, Shanghai, China, 764–780. https://doi.org/10.1007/978-3-642-17746-0_48
[19] Leslie G Valiant. 1990. A bridging model for parallel computation. *Commun. ACM* 33, 8 (1990), 103–111.
[20] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang (Eds.). ACM, Torino, Italy, 417–426. https://doi.org/10.1145/3269206.3271739
[21] Tian Wang, Hamid Krim, and Yannis Viniotis. 2013. A generalized Markov graph model: Application to social network analysis. *IEEE Journal of Selected Topics in Signal Processing* 7, 2 (2013), 318–332.
[22] Ye Wang, Qing Wang, Henning Koehler, and Yu Lin. 2021. Query-by-Sketch: Scaling Shortest Path Graph Queries on Very Large Networks. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, Virtual Event, 1946–1958. https://doi.org/10.1145/3448016.3452826
[23] Wencong Xiao, Jilong Xue, Youshan Miao, Zhen Li, Cheng Chen, Ming Wu, Wei Li, and Lidong Zhou. 2017. Tux$^2$: Distributed Graph Computation for Machine Learning. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, Aditya Akella and Jon Howell (Eds.). USENIX Association, Boston, MA, USA, 669–682. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/xiao

# Exploring the Utility of Graph Methods in HPC Thermal Modeling

Bruno Guindani
Department of Electronics, Information and Bioengineering
Politecnico di Milano
Milano, Italy
bruno.guindani@polimi.it

Martin Molan
Department of Electrical, Electronic and Information Engineering
Università degli Studi di Bologna
Bologna, Italy
martin.molan2@unibo.it

Andrea Bartolini
Department of Electrical, Electronic and Information Engineering
Università degli Studi di Bologna
Bologna, Italy
a.bartolini@unibo.it

Luca Benini
Department of Information Technology and Electrical Engineering
ETH Zurich
Zurich, Switzerland
lbenini@iis.ee.ethz.ch

## ABSTRACT

This work critically examines several approaches to temperature prediction for High-Performance Computing (HPC) systems, focusing on component-level and holistic models. In particular, we use publicly available data from the Tier-0 Marconi100 supercomputer and propose models ranging from a room-level Graph Neural Network (GNN) spatial model to node-level models. Our results highlight the importance of correct graph structures and suggest that while graph-based models can enhance predictions in certain scenarios, node-level models remain optimal when data is abundant. These findings contribute to understanding the effectiveness of different modeling approaches in HPC thermal prediction tasks, enabling proactive management of the modeled system.

## CCS CONCEPTS

• **Hardware** → **Temperature simulation and estimation**.

## KEYWORDS

High-Performance Computing, Graph Neural Network, Thermal Modeling

**ACM Reference Format:**
Bruno Guindani, Martin Molan, Andrea Bartolini, and Luca Benini. 2024. Exploring the Utility of Graph Methods in HPC Thermal Modeling. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3629527.3652895

## 1 INTRODUCTION

High-Performance Computing (HPC) represents a pinnacle of computational capability, harnessing the power of advanced hardware and software technologies to solve complex problems with unmatched speed and precision. The price for this unparalleled performance is the high hardware and operational costs, which become even more critical with the transition to exascale. The most crucial variable cost is the energy consumption of facility infrastructure, which is not directly linked to processing [9]. The cooling of the processing elements is the main contributor to this consumption along with its associated costs [3].

Intelligent thermal monitoring and prediction systems are being introduced to minimize the cooling expense and consequently reduce the variable cost of the HPC operation. These systems vary from macro-scale, predicting the power-usage efficiency of the entire data center in connection to weather conditions, to micro-scale, modeling the thermal dynamics of the processor of the single compute node. Accurate and reliable thermal prediction models would enable more efficient utilization of HPC computing systems, such as direct integration with the scheduler [3]. Energy-aware scheduling, such as the one proposed in [3], is already a well-explored concept in the HPC domain. During periods of high cooling costs, the scheduler aims to schedule fewer and fewer intensive compute jobs on the system.

Associated with the rise in performance, HPC systems have also exploded in complexity. Exascale and pre-exascale systems have up to tens of thousands of compute nodes, each consisting of CPUs and dedicated accelerators [9]. This explosion in complexity necessitates the transition from manual analysis and domain-driven models to the introduction of machine learning models. The most recent trend in machine learning-powered predictive models for HPC is the use of graph representation and Graph Neural Networks (GNNs) [6]. HPC systems are an ideal target for GNNs as there are multiple layers of connection between logical units (compute nodes), such as physical layout or job proximity.

In this work, we critically examine the utility of the graph processing approaches for the thermal prediction use case and compare it against the domain-driven per-node model. Based on this validation, we find the best node-level thermal prediction model that can be scaled to current and future pre- and exascale HPC systems. We perform the experimental evaluation on a publicly available dataset curated by the University of Bologna [4]. All proposed models are

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652895

also publicly available and form the basis for future exploration of graph- and non-graph-based predictive models for HPC systems. Our code is available at https://gitlab.com/ecs-lab/hpc-thermal.

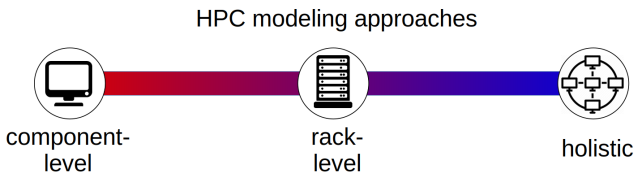



**Figure 1: Taxonomy of HPC modeling approaches.**

## 2 RELATED WORK

### 2.1 Machine learning approaches in HPC

There are two main approaches to building machine learning methodologies to support HPC systems: holistic and component-level models [8]. Each of them leverages the specific characteristics of HPC systems. Component-level approaches aim to build many different machine learning models such as neural networks, each tailored to a specific component or subsystem of the HPC system [7]. On the other hand, the holistic approaches aim to leverage large amounts of data produced by the HPC system to build a single, large model encompassing the whole HPC system with all its components and subsystems [1]. The midway point between room-level (holistic) and component-level models are models designed for a group of components, e.g, a set of nodes within the same compute rack [6]. We illustrate this taxonomy in Figure 1.

Component-level approaches are well-explored in the area of HPC modeling. For anomaly detection applications, it has been well-established that the best possible results come from training a separate self-supervised anomaly detection model for each compute node [7]. The common denominator of these approaches is that while the compute nodes are similar (e.g., they have the same hardware configuration), they nonetheless require models explicitly trained for those nodes. This is because each node experiences slight variations in hardware use, application utilization, and different cooling and thermal conditions. What is common across all the component-level approaches is that they use the same neural network structure, but the model for each node is trained from scratch. Using the same model structure for each of the component-level models is possible because the compute nodes in a compute room share the same hardware characteristics [7].

On the other side of the spectrum are the holistic modeling approaches (see [1, 7]). Instead of training many models with the same structure, they attempt to create a single model that provides predictions for all the components. Because of the data's large quantity and complexity, regular tabular modeling approaches cannot be utilized. Additional information in the form of a graph structure is introduced to train such models effectively. This graph structure often takes advantage of the fact that the HPC systems are organized in compute rooms, in which nodes are arranged in rows of compute racks. This physical layout can then be a basis for the graph representation of the room-level dataset. Graph-level models have proven useful for some problems, like anomaly anticipation, where they vastly outperform the component-level models [6].

### 2.2 Thermal modeling in HPC

In line with the taxonomy proposed in Figure 1, several thermal modeling approaches exist for the problem of compute node-level temperature prediction. These models, such as the one proposed in [2], belong to the component-level category. While there have been attempts [2, 10], [3] at building a holistic thermal model of the entire computing room, none of these approaches attempt to generate thermal predictions at the granularity of the individual compute node. Instead, existing holistic, room-level models only predict the average temperature of the entire computing room. However, for optimizing the energy efficiency of the HPC operations, as well as for energy-aware scheduling, more granular, node-level temperature predictions are needed [3].

Recent advancements in applications of graph processing methodologies, such as [6], suggest using Graph Neural Networks (GNNs) to build a holistic predictive model with a high resolution of predictions (for each individual node).

Motivated by the need for the node-level thermal prediction model, this work critically examines the component-level (node-level in our case) and the holistic modeling approaches. Specifically, it compares the same-structure, individually-trained approach (in line with [7]) against the graph-based approach inspired by [6]. To the best of our knowledge, this is the first work in the literature with a systematic focus on HPC temperature prediction via graph models.

## 3 METHODOLOGY

### 3.1 Dataset

The data used for our analysis comes from the publicly available M100 dataset [4]. It contains several features, also referred to as metrics, collected from the Tier-0 CINECA Marconi100 supercomputer over multiple years by the ExaMon HPC monitoring framework [3]. In particular, we focus on the April 2021 – September 2022 period, in which ExaMon collected all the metrics that are relevant to this work. We use this data in samples covering contiguous 15-minute time windows, each representing a snapshot of the HPC system at a certain period in time. The snapshots include, for each compute node, aggregations of data (such as average, minimum, maximum, and standard deviation) collected during the time window. The use of aggregations is necessary to have a time-uniform dataset since ExaMon samples different metrics at different frequencies. We define sample $t$ as the snapshot starting from timestamp $t$ and ending at 15 minutes after $t$. We refer to subsequent snapshots as sample $t + 1$, $t + 2$, etc.

We exclude samples that contain a proportion of missing values (NaNs) which is larger than 1%, a threshold we fixed after a comprehensive inspection of the data. This choice allows to skip snapshots with clusters of neighboring vertices with missing values, which would be difficult to impute. On the other hand, in the remaining eligible samples, we fill in NaNs with a *neighbor average* approach. That is, for a given missing value associated with a certain vertex $n$, metric $m$, and timestamp $t$, we collect the (non-missing) values of the $m$ metric of the neighbors of $n$ at time $t$, perform a weighted

average, and impute such average to the missing value. The weights used are the same as the corresponding edge weights in the graph model (see Section 3.3).

## 3.2 Prediction target

Given a snapshot associated with time $t$ and a compute node, our prediction target variable is the *future temperature increase* measured at the node outlet, i.e., the difference between the outlet temperature measured at time at time $t + 1$ and the one measured at time $t$. This corresponds to a 15-minute prediction horizon, as mentioned earlier. This time horizon enables almost real-time monitoring of the HPC system, allowing timely interventions by system administrators or automatic adjustments to the cooling system if using adaptive control strategies.

We choose to predict a temperature increase rather than the raw temperature data because it contains the truly relevant information that system administrators ought to know. A large raw temperature value does not necessarily suggest a hazardous state. External factors such as weather and seasonal conditions heavily influence such value, to the point where it is not uncommon to have an average 10°C difference between both winter and summer [2]. A sudden spike in temperature instead signals a potentially hazardous state of the system, especially using a short prediction time horizon.

## 3.3 Models

For our temperature prediction task, we propose three models, which represent three hypotheses having increasing complexity: 1) a Ridge linear regression model for each node; 2) a Dense Neural Network (DNN) model for each node; and 3) one Graph Neural Network (GNN) model for the entire Marconi100 room.

In the graph model, we represent the Marconi100 room with an undirected weighted graph whose structure is based on the physical layout of the room. We display this layout in Figure 2. Each dot

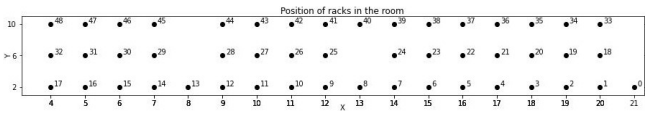



**Figure 2: Spatial coordinates (in meters) of Marconi100 racks.**

in the figure represents one of the 49 racks (IBM 7965-S42) in the room, each of which holds 20 compute nodes and is about 2 meters tall. The room therefore hosts a total of 980 compute nodes.

In our graph model, each vertex represents a compute node, with edges connecting it to its closest neighbors in all three spatial directions. Therefore, each vertex has at least 2 neighbors (for nodes in the corners of the room) up to 6 (for any node that is not near the sides of the room). This results in a total of 4782 weighted edges. Furthermore, edge weights are inversely proportional to the physical distance between nodes in the room. We refer to this representation as the *spatial graph* model.

We use this graph structure to perform regression using the Graph Convolutional Network (GCNConv) presented in [5]. We chose this network as it yields the best results in terms of prediction accuracy, according to preliminary experiments.

# 4 EXPERIMENTAL RESULTS

## 4.1 Experimental setting

The goal of our experiments is to assess the validity of the three models presented earlier. Specifically, for graph models, we seek to validate or reject the following two research questions:

- Does the data present a graph structure?
- Does a graph structure bring benefit to the prediction task compared to a per-node analysis?

Therefore, besides the comparison between per-node and graph models, we also conduct experiments with other graph mapping structures. Specifically, we also test a random connectivity matrix and a null connectivity matrix. In the first case, we sample random pairs of vertices and connect them through edges with randomly sampled weights. In the second case, we set all edge weights to zero, preventing the model from exchanging information between neighboring vertices.

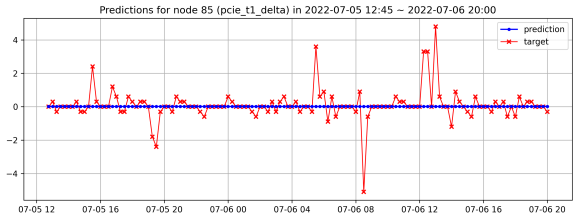
We split the collection of samples into a training set and a test set, composed of 80% and 20% of the total snapshots, respectively. We preserve the chronological ordering of snapshots during the split. This way, we simulate a real-world scenario in which we use past experience to predict the future behavior of the system. We use this split to train all models described in this work. We use the Mean Squared Error (MSE) as the loss function to minimize during training and as the validation metric for the evaluation of the test set. Relative metrics like the Mean Absolute Percentage Error (MAPE) are not suited for this prediction task, since the target variable is oftentimes very close to zero. For the graph models, we train each GCN in batches of size 20 for 10 epochs. Further details about the training process can be found in our code repository.

The baseline against which we compare all models is the trivial Last-Value Prediction (LVP), in which the prediction for a node's temperature at time $t + 1$ is equal to the temperature at time $t$. We use the temperature difference as the regression target, therefore the LVP is identically zero for all $t$. The LVP is often used as a reference benchmark for temperature in HPC settings since temperature is a slow-changing metric. In particular, the LVP represents a steady-state system approximation for the HPC system. Focusing on a performance comparison with the LVP means focusing on the meaningful temperature changes in the system.

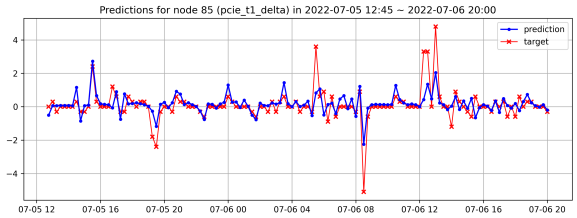
The random connectivity matrix described earlier is created by first fixing the same number of edges as in the spatial mapping (4782), then randomly sampling that many pairs of vertices. Each pair corresponds to a graph edge in the random mapping. Then, the weights for these edges are i.i.d. sampled from a uniform distribution on $(0, 10)$, which is a similar range to the original spatial mapping.

## 4.2 Empirical results

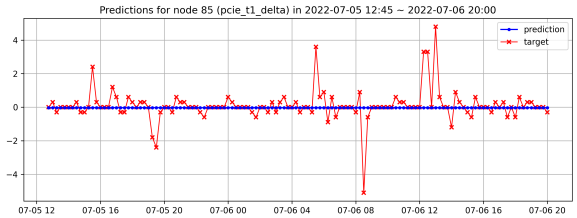
We will describe our process of improving our prediction models to improve their accuracy. The steps of this incremental process each refer to one part of Figure 3, in which we plot the true target values (in red) against the model predictions (in green) in the first portion of the test set, on a randomly chosen node. We will show that these steps lead to an overarching conclusion: *Simplicity is best.*



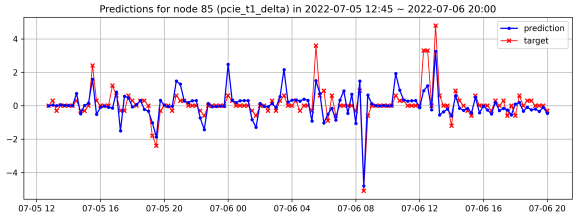

**(a) Spatial graph mapping: initial high-complexity model.**



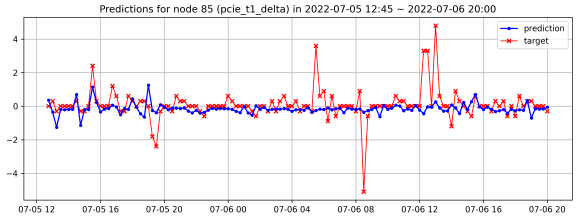

**(b) Spatial graph mapping: reduced model.**



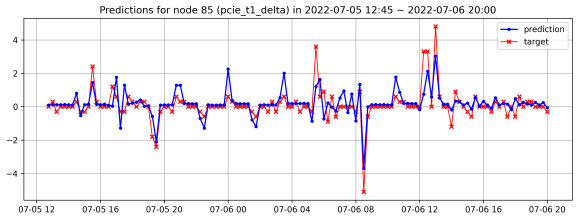

**(c) Node DNN model.**




**(d) Ridge linear model.**




**(e) Random graph mapping.**




**(f) No graph mapping.**

**Figure 3: Predictions for several models for node 85.**

*4.2.1 Model complexity.* We build our first GNN model as described in the earlier sections. In particular, we use as input all 416 aggregated node features available in the original dataset and use 10 hidden layers with regularly decreasing sizes. As shown in Figure 3a, such a model collapses to the trivial prediction of zero for all timestamps. This is also the average of all per-node temperature differences in the entire dataset, as well as the null constant value of the LVP baseline.

Since this model is unable to capture the trend of the target variable, we tried increasing the number of layers in the neural network. Still, the added complexity did not translate to an improvement in the model's predicting power. The result is nearly identical to the original model shown in Figure 3a. We then tried to head in the opposite direction by *removing* complexity from the model, in two different ways: by decreasing the number of hidden layers and by considering a smaller subset of all available features. In particular, out of the original 416 aggregated node features, we only choose to keep three that we deem the most relevant and essential. Two of these features are used twice, once representing information coming from snapshot $t-1$, and once from snapshot $t$. This results in five features being used in the GCN models. We list such features in Table 1. The underlying physical processes of heating suggest

| name | description |
|---|---|
| `pcie_avg_t-1` | average outlet temperature in snapshot $t-1$ |
| `pcie_t0` | outlet temperature at time instant $t$ |
| `total_power_avg_t-1` | average node power consumption in snapshot $t-1$ |
| `total_power_avg_t0` | average node power consumption in snapshot $t$ |
| `ambient_avg_t-1` | average inlet temperature in snapshot $t-1$ |

**Table 1: Reduced subset of node features.**

that given temperature information from time $t-1$ to $t$, as well as the total power consumption from time $t-1$ to $t+1$, we should be able to infer the temperature at time $t+1$. In Figure 3b, we can see that the changes bring positive benefits to the model, which is now capable of capturing the fluctuations of the next temperature difference. We may attribute the worse performance when using all available features to the fact that most of the removed ones are likely irrelevant when inferring temperature, therefore representing a source of noise.

Simplicity also has a positive impact concerning the GNN model complexity. Specifically, the best-performing model only has one hidden GCN layer (plus one input and one output layer). As mentioned, increasing the number of hidden layers does not improve the model performance, and can even have a negative impact when adding too many. We can explain this phenomenon by the determinism of the underlying physical process.

Simplicity also translates to a lack of complexity in the chosen GNN model. As previously mentioned, we tested several network models, and GCN [5] stood out as yielding the best prediction performance. This is one of the simplest models available in the literature (see, e.g., [11]).

*4.2.2 Per-node models.* We now build individual Dense Neural Network (DNN) models for each individual compute node, by using the same layer structure as the GNN, except with GCN layers swapped out for traditional fully connected layers. We also use as

| model | subplot | MSE |
|---|---|---|
| LVP | – | 2.260 |
| spatial graph (initial) | (a) | 2.261 |
| spatial graph (reduced) | (b) | 1.667 |
| node DNN | (c) | 2.261 |
| Ridge | (d) | 1.103 |
| random graph | (e) | 2.436 |
| no graph | (f) | 1.453 |

**Table 2: Average test-set Mean Squared Errors for several models.**

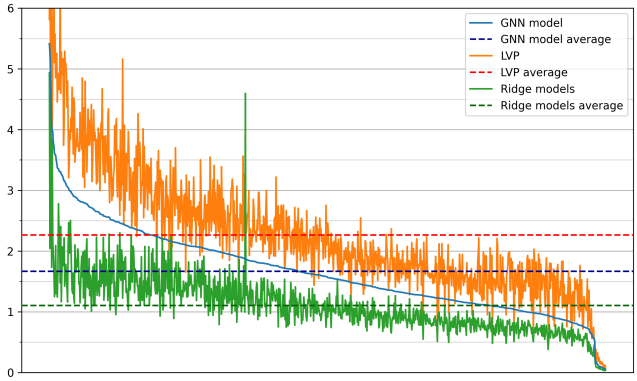
input the same reduced subset of the five features described earlier. However, as depicted in Figure 3c, this model also degenerates to a constant prediction. This result is consistent with our observation about complexity. Indeed, an entire neural network for an individual compute node is arguably more complex than a portion of a single graph network modeling all nodes at once – in other words, 980 independent DNNs bring more complexity than one GNN.

Finally, we attempt to simplify our assumptions even further by training one Ridge linear regression model for each compute node. Similarly to the DNN per-node model, we use the reduced subset of features. As exemplified in Figure 3d, not only does this model not collapse to a constant, but it also outperforms the best GNN model trained so far, i.e., the reduced one (Figure 3b). This is another piece of evidence suggesting that less complex models are best suited to this prediction task.

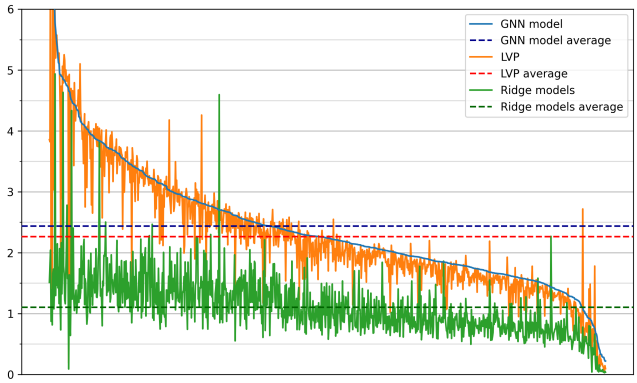
*4.2.3 Model comparison.* We show the full extent of the comparison between the GNN model and the linear model in Figure 4a. Specifically, each point on the blue line represents the average test-set MSE of the GNN predictions on an individual node. These points are sorted in decreasing order. The figure also shows the corresponding MSE for both the LVP baseline (in orange) and the Ridge linear model (in green), by keeping the same order of nodes used for the GNN errors. In other words, vertically aligned points refer to the same compute node. Finally, the horizontal dashed lines represent the average MSEs of the three models across all compute nodes. Figure 4a shows that Ridge models outperform the GNN model in nearly all individual nodes. Nonetheless, both models score lower errors than the LVP baseline.

*4.2.4 Graph structure validation.* We now report results for both the random and the zero-weight graph mapping. Their predictions on the chosen test-set window are displayed in Figures 3e and 3f, while the errors for all compute nodes are in Figures 4b and 4c, respectively. The random graph model does not degenerate to a constant either, but its predictions are less precise than the reduced spatial model – in fact, they turn out to be worse than even the trivial LVP baseline. This is most evident from Figure 4b: the curve of the GNN model error (in blue) lies above the LVP errors (in orange) for the majority of the nodes. On the contrary, the zero-weight graph model shows an improvement compared to the spatial model but still falls short of the linear regression model (as is clear from Figure 4c).

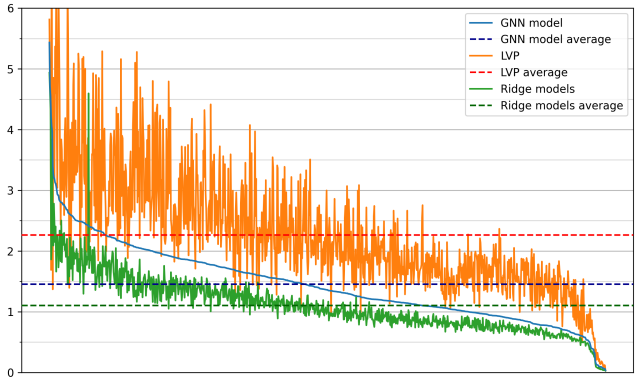
We show the average test-set MSEs of all models in Table 2. These figures are consistent with the rest of our analysis. The initial




**(a) Spatial graph mapping: reduced model.**




**(b) Random graph mapping.**




**(c) No graph mapping.**

**Figure 4: Sorted Mean Squared Errors for several models.**

spatial graph model (a) and the node DNN model (c) are very close to being constant models, thus they have the same error as the LVP baseline. The random graph model (e) performs worse than the baseline and the reduced spatial graph model (b) achieves worse performance than the zero-weight graph model (f), which in turn is worse than the per-node linear model (d).

*4.2.5 Graph vs per-node models.* These results lead us to the following considerations. The fact that the best models (in terms of

MSE) do not use graph information, shows that a graph structure is not needed to perform prediction on the M100 data, nor does a graph structure necessarily improve results in a regression setting. However, an inappropriate graph structure (such as the random one in (e)) can still influence the model's predictive capability in a negative way. This indicates that the data may still present graph-like patterns, which do not emerge when using a random graph structure.

## 5 DISCUSSION

This work critically examines the node-level and holistic modeling approaches for thermal prediction in HPC systems. Based on the experimental evaluation, the best-performing holistic approach is the spatial GNN which is trained to predict temperature changes in connection to the current temperature. It beats the baseline and achieves good prediction results. We also observe the importance of a correct graph structure: if the graph structure is shuffled, the prediction performance drops significantly, becoming even worse than the trivial baseline.

However, the holistic approach performs sub-optimally compared to the node-level approach. A simple node-level model based on domain-based feature engineering outperforms the graph model. This contrasts with the recent result in [6] on the same dataset, where graph-based models severely outperformed the node-level models on anomaly prediction tasks.

The difference between the anomaly prediction and the temperature prediction cases is the availability of labels. Anomaly prediction is an unbalanced classification task; per-node models do not have sufficient data to learn non-trivial predictions. In the temperature prediction case, however, the future temperature information is equally abundant in both the graph and node-level cases. The comprehensive study conducted in this paper thus suggests that when sufficient data is available, simple node-based models outperform more complex graph models.

Surprising results presented in this study nicely complement the current preliminary results on GNN applications in the HPC domain. While graphs are a powerful computational tool that unlocks the possibilities of fulfilling novel predictive tasks, their utility lies mainly in augmenting other models' poor or missing data, such as anomaly prediction cases. However, when data is abundant, the classic per-node and per-component modeling approaches still give the optimal results.

## ACKNOWLEDGMENTS

This research was partly supported by the HE EU Graph-Massivizer project (g.a. 101093202). This work was also supported by the Italian Ministry of University and Research (MUR) under the National Recovery and Resilience Plan (PNRR) and by the European Union (EU) under the NextGenerationEU project. Finally, we thank CINECA for their collaboration and access to their machines.

## REFERENCES

[1] Aksar, B., Zhang, Y., and Ates, E. e. a. Proctor: A semi-supervised performance anomaly diagnosis framework for production hpc systems. In *High Performance Computing: 36th International Conference, ISC High Performance 2021, Virtual Event, June 24–July 2, 2021, Proceedings 36* (2021), Springer, pp. 195–214.

[2] Ardebili, M. S., Bartolini, A., Acquaviva, A., and Benini, L. Rule-based thermal anomaly detection for tier-0 hpc systems. In *International Conference on High Performance Computing* (2022), Springer, pp. 262–276.

[3] Borghesi, A., Conficoni, C., Lombardi, M., and Bartolini, A. Ms3: A mediterranean-stile job scheduler for supercomputers-do less when it's too hot! In *2015 International Conference on High Performance Computing & Simulation (HPCS)* (2015), IEEE, pp. 88–95.

[4] Borghesi, A., Di Santi, C., Molan, M., Ardebili, M. S., Mauri, A., Guarrasi, M., Galetti, D., Cestari, M., Barchi, F., Benini, L., et al. M100 exadata: a data collection campaign on the cineca's marconi100 tier-0 supercomputer. *Scientific Data 10*, 1 (2023), 288.

[5] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations* (2017).

[6] Molan, M., Ahmed Khan, J., Borghesi, A., and Bartolini, A. Graph neural networks for anomaly anticipation in hpc systems. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering* (2023), pp. 239–244.

[7] Molan, M., Borghesi, A., Cesarini, D., Benini, L., and Bartolini, A. Ruad: Unsupervised anomaly detection in hpc systems. *Future Generation Computer Systems 141* (2023), 542–554.

[8] Ott, M., Shin, W., Bourassa, N., Wilde, T., Ceballos, S., Romanus, M., and Bates, N. Global experiences with hpc operational data measurement, collection and analysis. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)* (2020), IEEE, pp. 499–508.

[9] Shin, W., Oles, V., Karimi, A. M., Ellis, J. A., and Wang, F. Revealing power, energy and thermal dynamics of a 200pf pre-exascale supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2021), pp. 1–14.

[10] Taneja, S., Zhou, Y., and Qin, X. Thermal benchmarking and modeling for hpc using big data applications. *Future Generation Computer Systems 87* (2018), 372–381.

[11] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications. *AI open 1* (2020), 57–81.

# AutoGrAN: Autonomous Vehicle LiDAR Contaminant Detection using Graph Attention Networks

Grafika Jati
DEI Department, University of Bologna
Italy
grafika.jati2@unibo.it

Martin Molan
DEI Department, University of Bologna
Italy
martin.molan2@unibo.it

Junaid Ahmed Khan
DEI Department, University of Bologna
Italy
junaidahmed.khan@unibo.it

Francesco Barchi
DEI Department, University of Bologna
Italy
francesco.barchi@unibo.it

Andrea Bartolini
DEI Department, University of Bologna
Italy
a.bartolini@unibo.it

Giuseppe Mercurio
FEV Italia s.r.l.
Italy
mercurio_g@fev.com

Andrea Acquaviva
DEI Department, University of Bologna
Italy
andrea.acquaviva@unibo.it

## ABSTRACT

Extreme conditions and the integrity of LiDAR sensors influence AI perception models in autonomous vehicles. Lens contamination caused by external particles can compromise LiDAR object detection performance. Automatic contaminant detection is important to improve reliability of sensor information propagated to the user or to object detection algorithms. However, dynamic conditions such as variations in location, distance, and types of objects around the autonomous vehicle make robust and fast contaminant detection significantly challenging.

We propose a method for contaminant detection using voxel-based graph transformation to address the challenge of sparse LiDAR data. This method considers LiDAR points as graph nodes and employs a graph attention layer to enhance the accuracy of contaminant detection. Additionally, we introduce cross-environment training and testing on real-world contaminant LiDAR data to ensure high generalization across different environments. Compared with the current state-of-the-art approaches in contaminant detection, our proposed method significantly improves the performance by as much as 0.1575 in F1-score. Consistently achieving F1 scores of 0.936, 0.902, and 0.920 across various testing scenarios, our method demonstrates robustness and adaptability. Requiring 128 milliseconds on a AMD EPYC 74F3 CPU for the end-to-end process, our method is well-suited for an early warning system, outperforming human reaction times, which require at least 390 milliseconds to detect hazards. This significantly contributes to enhancing safety and reliability in the operations of autonomous vehicles.

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652896

## CCS CONCEPTS

• **Computing methodologies** → **Scene anomaly detection**; **Vision for robotics**; **3D imaging**.

## KEYWORDS

Advanced driver assistance systems (ADAS), autonomous vehicle perception systems, LiDAR Point Cloud reliability, LiDAR contamination and Noise detection, Graph Processing, Graph Attention Networks

**ACM Reference Format:**
Grafika Jati, Martin Molan, Junaid Ahmed Khan, Francesco Barchi, Andrea Bartolini, Giuseppe Mercurio, and Andrea Acquaviva. 2024. AutoGrAN: Autonomous Vehicle LiDAR Contaminant Detection using Graph Attention Networks. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3652896

## 1 INTRODUCTION

The LiDAR becomes one of the main sensors for the perception model in autonomous vehicles. LiDAR point clouds provide 3D information about the surrounding environment, offering an in-depth picture of the objects around it [12, 16]. Perception models such as object detection, object recognition, scene reconstruction, motion estimation, and path planning are essential. The integrity of LiDAR data influences the performance of perception models in autonomous vehicles. Failure of integrity can result in catastrophic errors or failures.

LiDAR data integrity is essential in edge case conditions, such as sensor cover contamination. While the sensor may be in good condition, its perception can be limited by unexpected situations affecting its surface or cover. Beyond severe weather conditions, the sensor cover's cleanliness can significantly limit perception

capabilities. This limitation is evidenced by a decreased object detection accuracy, as highlighted in a recent benchmark study [5]. This benchmark found that weather-related contamination, including rain, snow, and fog, reduces object detection performance. For instance, the PV-RCNN method achieved an Average Precision (AP) of 84.39 under clean conditions, which dropped to 51.35 with rain-affected data. That study assessed the impact of adverse weather conditions leading to particle accumulation on the sensor cover. In the same survey, using synthetic corruption data (Gaussian, uniform, and impulse noise only), object detection performance decreased by up to 20% of AP. On the other hand, sensor cover contamination can also result from external factors, such as water, dust, oil, mud, and other external sources.

The impact of these contaminants on the LiDAR lens represents a major threat to sensor data integrity. This emphasizes the need for a contaminant detection technique as an early warning before executing object detection. Contaminant detection is also crucial for triggering automatic cleaning procedures if contaminants are present. Perception anomalies remain a hot topic nowadays, as summarized in a survey of several techniques for anomaly detection [2]. However, research in contaminant detection still needs to be improved due to the challenge of scarcity of real-world LiDAR-contaminated datasets [14].

A key attribute of LiDAR point clouds is the intensity of each point, which reflects the return strength of a laser beam after striking an object. Different materials reflect varying intensities and an object's distance can diminish this intensity [1]. Typically, the longer the distance from the sensor to the object, the lower the object's intensity. Thus, detecting contaminants becomes challenging when relying solely on intensity values in dense or sparse point clouds, as contaminant presence can be confused with large distance points. Thus, a distance and environment-invariant contaminant detection model is highly needed for real-world environments.

In this paper, we employ graph representation for contaminant detection with the following contributions: i) We are the first to tackle LiDAR contaminant detection using graph representation by comparing different approaches. ii) We improve state-of-the-art approaches in terms of accuracy and tested in various contaminant types from various environmental settings. In particular, detecting contamination is challenging due to LiDAR sparsity. LiDAR point clouds at longer distances are more sparse than closer ones. To tackle this sparsity, we propose a voxel-based graph transformation to deal with contaminated LiDAR point clouds.

Specifically, we employ a Graph Neural Networks (GNNs)-based method that treats LiDAR points as connected nodes in a graph. GNNs are specialized neural networks designed for graph-structured data, wherein nodes represent entities and edges represent relationships. They excel in capturing spatial information by aggregating data from neighboring nodes, enhancing their understanding of relationships and dependencies within the graph. In a LiDAR point cloud, where an object is present, the points would be denser and contain more information than in a no-object area. This leads us to employ graph attention networks. Additionally, we deal with large regions of the environment and high-resolution LiDAR, which generates a large number of points. Processing all points as nodes in the graph increases computation time, so representing some points as a single node helps reduce the number of nodes. This approach can be defined as point voxelization. Finally, we introduce a graph representation of voxel point clouds using graph attention networks (voxel-GAT) for contaminant detection on autonomous vehicles, namely AutoGrAN.

The proposed method efficiently represents sparse LiDAR data and outperforms the state-of-the-art approach regarding classification performance and computational efficiency. We utilize three datasets, namely *"5m," "10m," and "20m,"* to reflect variations in location, distance, surrounding objects, and contaminant sources. Our approach employs a cross-data validation scheme, creating three models based on each dataset and testing them using a different dataset. Compared to the 3D CNN-based detection method, our proposed method improves the F1-score from 0.778, 0.850, and 0.840 to 0.936, 0.902, and 0.920, respectively, across six possible train-test dataset scenarios.

In terms of computational performance, our proposed method achieves competitive end-to-end processing times, from graph construction until obtaining contaminant results, all within an average of 128 milliseconds per point cloud frame running on CPU AMD EPYC 74F3, single core CPU clock speed 3.2G. This efficiency establishes the proposed method as a promising candidate for an early warning system in autonomous vehicle downstream tasks. As indicated in [10] and [22], the system has to complete the end-to-end processing within a latency of around 100 milliseconds to be effective. Additionally, considering human reaction times, which range from 390 to 600 milliseconds for detecting incoming hazards [13] [7], the proposed method's computation times is a significant advantage. It delivers promising results in contaminant detection for LiDAR on autonomous vehicles, demonstrating high accuracy, robustness, ease to train, and low computational overhead.

## 2 RELATED WORKS

### 2.1 LiDAR processing for automotive applications

The perception model is a crucial input for the decision-making and planning of an autonomous vehicle. The research was carried out to increase each perception model's accuracy, robustness, and generalization, e.g., in [9],[21]. To achieve this goal, the development of the Autonomous vehicle model needs to pay attention to various aspects ranging from research to development. In more detail, an AI model's development depends on the data, algorithm, and deployment aspects.

In the paper by Wang et.al [20], it is mentioned that accuracy and robustness are important factors. High accuracy is certainly needed, considering that automotive vehicles require correct decision-making because they involve life-and-death situations. Errors in perception can cause decision-making errors, which can have fatal consequences. Robustness concerns the AI model's ability to perform well in every situation, dynamic environment, dynamic location, and object surrounding. We need to realize that there will be situations and scenarios of uncertainty in the real world. The sensor properties also influence the AI model, especially the perception model, which must be robust to all sensor conditions [3]. The perception model must be robust when deployed and tested in different environments, weather, and traffic situations [25].

Apart from accuracy and robustness, the perception model needs to consider ease of training. Autonomous vehicles must be usable under all conditions, making data collection a significant challenge in terms of both cost and time. The AI training process must be feasible with limited data. We lack datasets for training in every possible situation, including various adverse traffic conditions as well as challenges posed by weather and sensor contamination [24]. Efforts have been made with data degraded by simulators. However, simulating all scenarios, especially unusual ones (edge cases and anomalies), proves difficult. Formal verification using mathematical models can be applied to the created models, but scalability is limited, especially for complex systems like autonomous vehicles. Therefore, every AI model developed needs to undergo real-world testing to validate performance and identify new scenarios that may not have been previously considered. Real-world testing allows for the validation of AI performance and the discovery of previously unrecorded scenarios.

The last aspect to consider regarding AI models is low computational overhead. Autonomous vehicles require fast decision-making, where the perception model serves as input for the reinforcement learning agent in the autonomous vehicle's decision-making process. Therefore, it is crucial to maximize the speed of the subprocesses [17] [23]. Finally, developing LiDAR processing methods should be prioritized to meet the requirements of high accuracy, robustness, ease of training, and low computational overhead.

## 2.2 LiDAR point cloud transformation for contaminant detection

Driven by applications in autonomous driving, several LiDAR point cloud processing approaches have been developed. These can be categorized based on the type of data representation: 1D, 2D, 3D, and graph. The 1D approach creates a representative vector from the original 3D data, upon which classical machine learning techniques for tabular data are applied.

In research aligned with contaminant classification, the use of 1D transformations was introduced by Heinzler et al., who aggregated features from point clouds such as $(x, y, z)$ for the cartesian and $(r, \theta, \varphi)$ for the spherical coordinates, echo number, intensity, and echo pulse width. In their study, Heinzler et al. attempted to classify weather conditions as clear, rainy, or foggy on LiDAR point clouds, achieving the best accuracy of 97.14% using SVM and KNN [6]. Similarly, the application of 1D transformation and KNN for classifying rain, fog, and snow was proposed by Rivero et al. [19]. However, Rivero placed the LiDAR sensor in a static position. Furthermore, both [19] and [6] did not address the classification of sensor cover contamination.

Another approach involves 2D transformation, which constructs a 2D depth and intensity image from the LiDAR point cloud. James et al. initiated the classification of sensor cover contamination using 2D transformations [8]. They transformed a 2D image of LiDAR data as input for a 2D CNN to classify contaminants like dirt, salt, and frost. The 2D CNN achieved promising results, reaching an accuracy of around 80% in classifying between clean and dirty conditions using a front-view transformation of LiDAR data. However, the data used in [8] was collected from a statically positioned sensor without any objects in front of the acquisition sensor and with the sensor cover fully contaminated. Therefore, it may not accurately reflect real-world environmental conditions.

The 2D transformation inevitably reduces spatial information from the original 3D point cloud. LiDAR processing that uses 3D data directly often employs voxelization techniques. This technique distributes the point cloud coordinates while preserving as much spatial information as possible. The application of 3D voxelization includes its use in PV-RCNN, which leverages 3D convolution as a backbone for 3D object detection [4]. However, the proposal to use 3D convolution for contaminant detection has not yet been established. Nonetheless, we have considered 3D as one of our baseline methods for contaminant detection.

The latest advancement in LiDAR processing methodologies is the introduction of graph processing networks, which aim to represent sparse LiDAR sensor data as graphs. Graph convolutional networks were then applied to this graph-represented data. Graph convolutions can be significantly more efficient than 2D and 3D convolutions on sparse tensors, as they avoid unnecessary iterations over zero elements.

The use of graphs for point cloud analysis was initiated by Point-GNN, which successfully employed a Graph Neural Network to process point clouds for object detection [15]. However, Point-GNN's limitation lies in its inability to capture the global context of the environment, especially in large settings. Subsequently, graph attention mechanisms were developed for object detection from raw LiDAR data [18]. Recent research has utilized Graph Attention Networks for 3D object detection [11]. These methods have inspired the use of voxels and attention in graph classification. A simple and highly accurate network is needed for contaminant detection. To the best of our knowledge, no study has leveraged a graph-based approach for contaminant classification research.

# 3 METHODOLOGY

This paper proposes point cloud contaminant detection as a graph classification problem. Due to the lack of a dedicated dataset for this task, we collected the LiDAR data ourselves, which includes varying levels of location, distance, sparsity, and various types and levels of contaminations (for details, refer to subsection 3.1). Recognizing the sparsity of the data in our dataset, we decided to use graph attention networks (GATs), which utilize the attention mechanism to handle data sparsity effectively. A LiDAR point cloud comprises up to 2 million points; if a single node represents a single point, it causes high computational costs. To address this challenge, we propose voxelization, which involves dividing the 3D space of point clouds occupied by some points into small cubic volumes called voxels so that a single node will represent several points. This reduction in the number of nodes significantly reduces computational time. We then create the graph structure from the voxelized data (detailed in subsection 3.2) to be passed into the GAT network (described in subsection 3.3). The network outputs a single global representation of each input voxel graph as a binary classifier, aiming to detect contaminant presence or absence.

## 3.1 Cover Contamination LiDAR Dataset

The point cloud data were captured using a test-bed car equipped with a LiDAR sensor, specifically the RS-Ruby from Robosense,

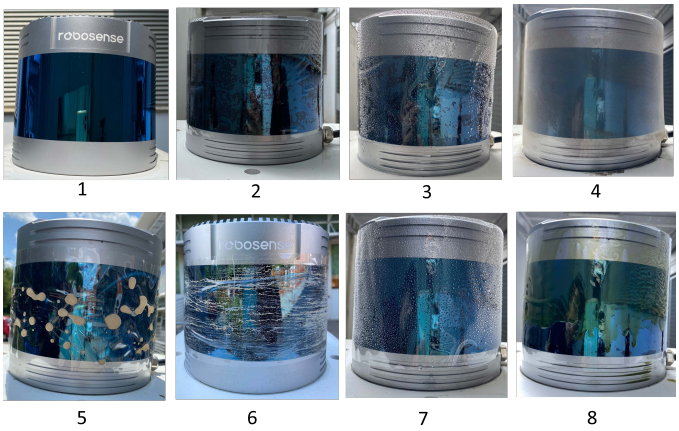



**Figure 1: Applying contaminant on LiDAR cover: (1) clean,(2) clean-cover, (3) tap water, (4) dust, (5) mud drop, (6) mud uniform, (7) salt water, (8) lubricant oil.**

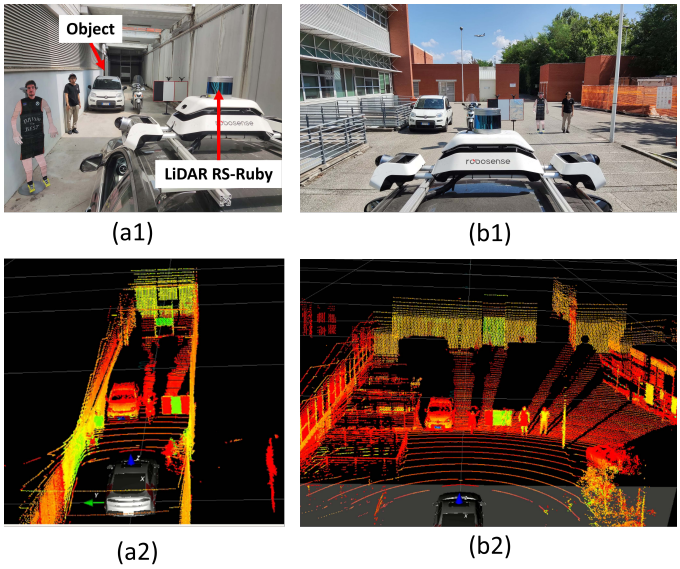



**Figure 2: Contaminated LiDAR data acquisition using testbed-car, contaminated LiDAR, and surrounding object: (a1) data $5m$ in an underground narrow hallway, (a2) raw LiDAR acquired in $5m$, (b1) Data $10m$ data outdoor parking area, (a2) raw LiDAR acquired in $10m$.**

rated at level L4+ or considered High Driving Automation. The LiDAR sensor has specifications of 128 signal beams, maximum 250 meters range, Horizontal resolution $0.2^o$, minimum $0.1^o$ Vertical Angular Resolution, Horizontal FoV $360^o$, Vertical FoV $40^o$, 10 Hz frame rate.

This contaminant detection model differentiates between two classes: clean and contaminated. The clean class includes two conditions: (1) the sensor is free from contaminants, and (2) the sensor is covered with a clean-transparent plastic cover. The contaminated class covers various types of contamination, including water, dry and wet mud, dust, salt water, and engine lubricant/oil on top of clean-transparent plastic cover. The application of contaminants is shown in Figure 1. Each type of contaminant is applied separately in different experiments. Each contamination has low, middle, and high levels, corresponding to varying amounts of spray used when applying the contaminant to the sensor cover. For example, 'low' corresponds to one spray application, 'mid' to three, and 'high' to five. This approach aims to reflect the diverse characteristics of real-world contaminants. To simulate objects encountered on the road, we covered several objects, such as cars, pedestrians, motorbikes, whiteboards, and aluminum foil.

To evaluate high generalization capabilities with high accuracy and robustness, this research utilizes three distinct datasets representing varied environmental settings: the $5m$, $10m$, and $20m$ datasets. The $5m$ collected dataset is of a passage to an underground parking lot, where the hallway is below ground level and is confined by walls on both sides. In this dataset, the LiDAR is positioned with the car object around 5 meters ahead, surrounded by other objects. The $10m$ and $20m$ datasets have data collected in the exact location. The location is an outdoor parking lot with wider spatial dimensions compared to the $5m$ data. In the $10m$ dataset, the target object is positioned around 10 meters from the LiDAR, while in the $20m$ dataset, the target object is placed at around 20 meters. Figure 2 depicts our data acquisition setup.

We select a specific area in front of the LiDAR sensor to ensure the effect of contamination. This area encompasses all three dimensions for the training and testing datasets, with the LiDAR-equipped car serving as the reference coordinate (Point 0,0,0). We take the area of interest where target objects are placed in our environment. So, along the x-axis, we choose a span of 80 meters from the reference point (0 to 80 meters). For the y-axis, we select an area of 15.5 meters (-5.5 to 10 meters in coordinate); for the z-axis, we take all the return points. The same area of interest cropping procedure is applied to all three datasets. The $5m$ dataset typically yields around 140,000 points per point cloud, in contrast to the $10m$ and $20m$ datasets, which generally yield around 70,000 points per point cloud. The higher point density in the $5m$ dataset is due to the closer distance of the LiDAR to the target object, resulting in denser point clouds.

## 3.2 Graph Construction

To construct graphs, we first apply voxelization to reduce the number of points to avoid computational overhead. Each point cloud is converted into a three-dimensional voxel grid. The number of voxels differs for each point cloud depending on the original structure or environment in real-world situations.

In voxelization, we employ the concept of average pooling to maintain the primary feature representation of the point cloud, which, in the case of LiDAR, is the point cloud structure and intensity of each point. Each voxel computes and retains the average intensity of the points it encapsulates, thus maintaining spatial information of the original point cloud. Applying a 3D convolutional layer directly on 3D voxels might make our predictions less accurate because it processes every voxel, even the ones without any points. To address this issue, we propose converting voxels into graphs. In this graph representation, each node corresponds to

a voxel containing points and the edges to connect voxels closest to each other in three dimensions. This approach aims to enhance prediction accuracy by focusing exclusively on voxels containing relevant information and their immediate spatial relationships. The edges for each graph are created in a three-step process, as follows:

(1) Point sorting: The initial step is the most important of the whole process. We employ a QuickSort algorithm to sort the points according to their coordinates in the three dimensions $X, Y, Z$ separately. For $n$, the number of points along a dimension, the algorithm will have a $O(n \log n)$ complexity. Since we have it in three dimensions $(z, y, z)$, the overall complexity is $O(n \log n)$.
(2) Edge construction: This process considers adjacent points from the sorted lists in each dimension. Each point in every dimension can have at most two adjacent points, except for the first and last elements on the list. This results in approximately $2n$ edges for each dimension, totaling $6n$ across all three dimensions. This process has a $O(n)$ complexity. The list of edges is then saved in an edge index.
(3) Edge directionality: Contaminants primarily influence the intensity of the reflected LiDAR beams and the structure of objects rather than introducing directional effects. Therefore, in our graph construction, edges play a crucial role in capturing point distribution for each node according to its neighbor but do not necessitate bi-directionality. Therefore, the constructed graphs have un-directed edges. We utilize the coalesce function from the pytorch-geometric library to remove duplicate edges. In principle, this process would require $O(k \log k)$ where $k$ is the total number of edges before removing duplicates. In the worst case, $k$ can reach $6k$ for all three dimensions, so removing and looping a list of edges can require $O(k)$ in the worst case. However, since $k$ depends on $n$, this complexity can also be considered as part of $O(n \log n)$ because our proposed construction graph is dominated by sorting operations.

We create a *Data* pytorch-geometric library object representing a graph object. We select all three coordinates and the beam's intensity as a feature vector for each instance of a voxel. The edges we created earlier become the *edge_index* property of the *Data* object, which forms the adjacency matrix of nodes. The proposed method will construct a graph with a different number of nodes and edges according to the original input point cloud structure.

## 3.3 Graph Attention Networks

We construct a deep learning model that employs Graph Attention Networks (GATs). In LiDAR data acquisition, areas with objects often yield dense point clouds, resulting in sparsity within the well-suited data for GATs. The GATs efficiently handle sparse data by representing it as a graph, with each point as a node. Leveraging attention mechanisms, they focus on relevant subsets of nodes, adapting to the local structure of the graph. These features empower GATs to excel in classifying contamination within point clouds generated by LiDAR, making them a powerful tool for our task. Figure 3 outlines the proposed GATs model for contaminant classification, followed by table 1, which details the number of parameters in the proposed model. The proposed architecture comes from empirical experiments to get the minimum parameters with the highest F1 score.

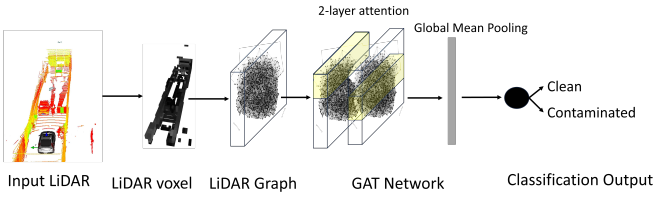



**Figure 3: The Proposed LiDAR Contaminant Detection based on Graph Attention Networks.**

**Table 1: The Proposed network for LiDAR Contaminant Detection.**

| |
|---|
| conv1.att_src: 64 |
| conv1.att_dst: 64 |
| conv1.bias: 64 |
| conv1.lin_src.weight: 256 |
| conv2.att_src: 2 |
| conv2.att_dst: 2 |
| conv2.bias: 2 |
| conv2.lin_src.weight: 128 |
| Total Parameters: 582 |

The model uses two GAT layers (GATConv) to process node features in the graph, generating an adaptive representation of each node based on its local context. The first layer employs multi-head attention mechanisms to enrich and diversify feature representations with four heads. Each node's features pass through this first GAT layer (conv1), followed by an ELU activation for non-linearity. The second layer (conv2) reduces the output feature dimension to one for each node. GAT's attention mechanism assigns varying weights to node neighbors, enhancing relationship capture. Since our task is graph classification, we then pass the graph through a (global mean pooling) aggregation function to get a representation of all the node features of the graph into a single value for binary classification: clean or contaminated. The model's output, the Log_softmax applied to this value, yields a probability distribution for clean and contaminated classes. Training employs the Adam optimizer with $lr = 0.005$ and $weight\_decay = 5e-4$.

# 4 RESULTS

## 4.1 Setting Experiments

*4.1.1 Dataset Preparation.* The numbers of contaminated class instances are 360, 328, and 364 for the $5m$, $10m$, and $20m$ datasets, respectively. The contaminated class combines various contamination sources explained in 3.1. While 360 instances represent the clean class across all datasets, we consider the data balanced. Each point cloud instance contains between around 70,000 and 140,000 points. Our proposed method transforms each point cloud instance into a graph, resulting in node-edge pairs: (n)1361-(e)3879 for the $5m$ dataset, (n)2209-(e)6359 for the $10m$ dataset, and (n)2514-(e)7242 for the $20m$ dataset. Training and testing are conducted using: CPU

AMD EPYC 74F3 24 core 48 threads, single core CPU clock speed 3.2G, single GPU NVIDIA GeForce RTX 3090 with 24265MiB, and RAM 250G DDR4.

To demonstrate the robustness of the contaminant detection method, we will conduct a cross-environment between training and testing data. Thus, three models will be developed, each trained on data from an environment, for example, 5$m$, and then tested on data from the other environments, 10$m$ and 20$m$, and vice versa. Utilizing three different datasets captures the dynamic aspects of the point cloud that are close to real-world conditions. This approach will also highlight the model's generalization and transferability capabilities. Therefore, even with limited training data, the detection model can perform effectively across different environmental settings.

*4.1.2 Baseline Method.* To demonstrate the complexity of contaminant detection in dynamic situations, we will compare classification models ranging from manual feature engineering to the most straightforward transformations in 1D and 3D. The point cloud is transformed into 1D data by extracting information such as $x, y, z$ coordinates, and intensity values, resulting in seven features: minimum, maximum, mean, standard deviation, and percentiles at 25, 50, and 75. This process generates 28 1D features that represent the statistical distribution of the point cloud. Subsequently, we compare various machine learning algorithms, including Logistic Regression, Multilayer Perceptron, Support Vector Machine, Naive Bayes, k-Nearest Neighbors, Decision Tree, and Random Forest. Additionally, we compare these methods to another baseline, namely the 3D approach, which preserves the spatial information from the point clouds. In the 3D approach, voxelization is performed, and the model is constructed using a 3D Convolutional Neural Network (3D CNN).

## 4.2 Contaminant Detection

The performance of contaminant detection is analyzed in terms of F1-score, graph construction time, and inference time across all possible scenarios. Table 2 evaluates the performance of various machine learning algorithms in detecting contaminants, measured by F1 scores, at three different data scales (5$m$, 10$m$, and 20$m$). The models compared include well-established classification techniques such as Decision Tree, Logistic Regression, Naive Bayes, and Support Vector Machine, as well as more modern approaches like Multi-Layer Perceptron, K-Nearest Neighbors, 3D Convolutional Neural Network, and the proposed method, namely Voxel-GAT.

From the data presented in Table 2, we observe that shallow learning models, such as Decision Trees (DT), Logistic Regression (Logit), and Naive Bayes (NB), are not able to perform adequately in contaminant detection tasks, with their average F1 score being below 80% across all tested data scales. This suggests that shallow learning methods may lack the capacity to capture and model complex relationships in three-dimensional spatial data, often involving nonlinear interactions and patterns that are difficult to separate. This limitation is likely because these models do not exploit the spatial structure in the data and tend to have more superficial feature representations, resulting in lower performance in tasks that require a deep contextual understanding. On the other hand, models such as 3D CNNs show improved performance compared to traditional models, supporting the theory that three-dimensional spatial data structures are inherently better suited to tasks with spatial characteristics.

The proposed method, Voxel-GAT, stands out in this comparison by demonstrating high F1 scores across all data scales, indicating its effectiveness in detecting contaminants with high consistency. This suggests that Voxel-GAT can capture complex spatial relationships in the data, an essential aspect of tasks that detect patterns or anomalies in three-dimensional spatial data. Voxel-GAT excels in ensuring distance-environment invariance, which is critical for explaining differences between the trained and tested data. As previously described, the 10$m$ data set is similar to the 20$m$ data set, whereas the 5$m$ data set differs regarding the location and distance of surrounding objects. The proposed model is more robust across different scenarios, being trained on 5$m$ and tested on 10$m$ and 20$m$ data, and vice versa. For the cross-train-test dataset, we have six possible scenarios, which, scenarios 1,2,3,4,5 and 6. All of the scenarios introduced are in table 2. Based on the table, scenarios 3 and 5 are trained on 10$m$ or 20$m$ and still performing well on the 5$m$ test data—a result not achieved by the baseline methods. The consistently superior performance of Voxel-GAT across all data scales confirms its adaptability and robustness in handling variations in data size and complexity, making it a promising choice for real-world contaminant detection applications.

The proposed method demonstrates outstanding performance, as reflected in the confusion matrix shown in Figure 4. We examine six Voxel-GAT confusion matrices for each scenario, corresponding to the scenario numbers in Table 2. Based on the confusion matrix displayed in Figure 4, it is evident that the Voxel-GAT model generally experiences a low number of False Negatives, with no instances of False Positives, except in Scenario 4. In this scenario, where the model was trained with 10$m$ data and tested with 20$m$ data, several cases of False Positives were observed. A closer inspection of the case data for each misclassification, detailed in Table 3, reveals that errors occurred with data originating from various sources of contaminants, namely water, uniform mud, mud drop, dust, and salt. Interestingly, misclassifications predominantly happen in cases of low contamination, where the contamination level is the lowest. An exception is observed with mud drops, where misclassifications occur at low, mid, and high contamination levels. This phenomenon is understandable, given that mud drops cover the sensor at specific locations with a volume of contaminant that is insignificant compared to the total sensor coverage.

The false positives identified in Scenarios 4 and 6 were attributed to the clean cover, which was mistakenly classified as contamination. This issue is not particularly problematic due to the nature of the cover installation. Contaminants were applied to the protective cover surrounding the LiDAR to safeguard the experimental (and expensive) LiDAR hardware. This cover introduces noise into the LiDAR image and sometimes acts as an anomaly/contaminant. Notably, the contaminant classification approach consistently recognized clean LiDAR data (without the cover) as non-contaminated. The instances involving the cover are the primary cause of false positives in Scenarios 4 and 6.

However, the problem with the cover data does not invalidate the robustness of the proposed contaminant detection model. If we focus only on the critical anomalies of middle and high contamination, we still observe that we correctly classify most cases. As only

**Table 2: Evaluation result of all cross train and test data scenario for contaminant detection from baseline and the proposed method Voxel-GAT. The number reported is F1-score.**

| No. Scenario | Model | Testing Data | 1D DT | 1D Logit | 1D MLP | 1D NB | 1D RF | 1D SVMrb | 1D KNN | 3D CNN | Voxel-GAT |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $5m$ | $10m$ | 0.308 | 0.308 | 0.308 | 0.308 | 0.308 | 0.711 | 0.308 | 0.780 | 0.918 |
| 2 | $5m$ | $20m$ | 0.336 | 0.336 | 0.336 | 0.336 | 0.336 | 0.728 | 0.336 | 0.777 | 0.954 |
| | **average F1** | | **0.322** | **0.322** | **0.322** | **0.322** | **0.322** | **0.719** | **0.322** | **0.778** | **0.936** |
| 3 | $10m$ | $5m$ | 0.676 | 0.512 | 0.336 | 0.333 | 0.526 | 0.486 | 0.720 | 0.733 | 0.889 |
| 4 | $10m$ | $20m$ | 0.343 | 0.349 | 0.825 | 0.336 | 0.971 | 0.936 | 0.900 | 0.968 | 0.915 |
| | **average F1** | | **0.509** | **0.430** | **0.580** | **0.334** | **0.748** | **0.711** | **0.810** | **0.850** | **0.902** |
| 5 | $20m$ | $5m$ | 0.361 | 0.333 | 0.336 | 0.333 | 0.361 | 0.381 | 0.568 | 0.733 | 0.876 |
| 6 | $20m$ | $10m$ | 0.871 | 0.833 | 0.955 | 0.308 | 0.977 | 0.940 | 0.977 | 0.948 | 0.964 |
| | **average F1** | | **0.616** | **0.583** | **0.645** | **0.320** | **0.669** | **0.660** | **0.772** | **0.840** | **0.920** |

these examples cause real problems in both autonomous driving applications and perception systems, they should be the focus of the proposed work. In practical applications in production-ready hardware, there would also be no problem with the cover and contamination as the LiDAR will operate without any additional plastic cover.

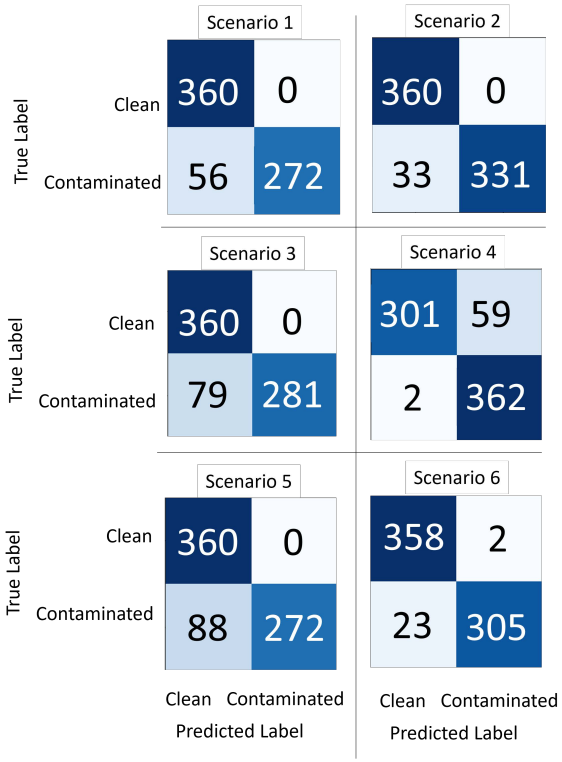



**Figure 4: Confusion matrix of the proposed method in all scenarios.**

Regarding computational performance, Table 4 compares the inference time of the baseline methods and the proposed method when running on both CPU and GPU. The 'CPU' column displays the inference time on the CPU, while the 'GPU' column shows the inference time when the process is executed on the GPU. Although the shallow machine learning methods are generally faster, their F1-score performance is significantly lower than Voxel-GAT.

**Table 3: Mis-classification cases of the proposed method Voxel-GAT.**

| No. Scenario | False Positive | False Negative |
|---|---|---|
| 1 | - | water:low; mudUniform:low,mid; dust:low; salt:low |
| 2 | - | water:low; mudUniform:low |
| 3 | - | water:low; mudDrop:low,mid,high; dust:low; salt:low |
| 4 | cover | water:low |
| 5 | - | water:low; mudDrop:low,mid,high; dust:low; salt:low |
| 6 | cover | water:low; mudUniform:low; salt:low |

**Table 4: Inference time of baseline method and proposed Voxel-GAT, running on CPU and GPU.**

| Method | CPU (in second) | GPU (in second) |
|---|---|---|
| DT | 5.44E-07 | x |
| Logit | 1.20E-06 | x |
| MLP | 6.82E-06 | x |
| NB | 5.46E-07 | x |
| RF | 1.62E-06 | x |
| SVMrb | 1.44E-05 | x |
| KNN | 1.46E-04 | x |
| 3D CNN | 6.10E-04 | 4.03E-05 |
| Voxel-GAT (Proposed) | 4.99E-03 | 7.91E-05 |

For the proposed method, the inference time on the CPU is recorded at 4.99e-03 seconds. On the GPU, the inference time reduces to 7.91e-05 seconds, indicating that the method benefits significantly from the parallel processing capabilities of the GPU. In comparison, the baseline 3D CNN has an inference time of 6.10e-04 seconds on the CPU and 4.03e-05 seconds on the GPU. This means that 3D CNN is about 1.96 times faster on the GPU than Voxel-GAT.

An early warning system should not only account for inference time but also include preprocessing time. The time required for preprocessing is 0.010 seconds, 0.129 seconds, and 0.128 seconds for 1D, 3D, and graph data, respectively. While 1D processing is faster, the time taken for voxel and graph creation is not significantly

longer, on average. When combining preprocessing with inference time, the total end-to-end time from raw data to detection result does not significantly increase. The time to create a graph is approximately 0.128 seconds or 128 milliseconds. However, considering the trade-off between the F1 score and inference time, Voxel-GAT is the preferable choice, offering a much higher F1 score than 3D CNN with a competitive end-to-end computational process.

## 5 CONCLUSIONS

The LiDAR contaminant detection method for autonomous vehicles has been developed. We compared all transformation methods, from 1D and 3D, to a developed graph-based transformation. The proposed method is superior in all experimental scenarios, including cross-environments and contamination, on real-world LiDAR sensors installed on test-bed vehicles.

The proposed method achieves an average F1-score above 0.902, with the graph processing time from creation to detection under around 128 milliseconds. We utilize a compact architecture containing two Graph Attention Networks layers capable of processing point cloud inputs of dynamic sizes. Ultimately, the proposed model has been proven to effectively classify cleanliness or contamination in complex and extreme LiDAR point clouds. This initiates further research into designing near-sensor LiDAR processing methods that are low-cost and sustainable using a graph processing approach.

## ACKNOWLEDGMENTS

This research is supported by EU through the National Recovery and Resilience Plan (NRRP) Mission 4, Component 2, Investment 3.3 (g.a. DM 352/2022). We also thank FEV Italia s.r.l. for the collaboration, access to their hardware, and help in experimental work. This research was supported in part by the UAE Technology Innovation Institute (TII) through the Zero-Trust project. This research was also partly supported by the Graph-Massivizer project (g.a. 101093202). We also thank to Giammarco Cecchini for helping during data acquisition.

## REFERENCES

[1] Csaba Benedek, Andras Majdik, Balazs Nagy, Zoltan Rozsa, and Tamas Sziranyi. 2021. Positioning and perception in LIDAR point clouds. *Digital Signal Processing* 119 (2021), 103193. https://doi.org/10.1016/j.dsp.2021.103193

[2] Daniel Bogdoll et al. 2022. Anomaly detection in autonomous driving: A survey. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 4488–4499.

[3] Krzysztof Czarnecki and Rick Salay. 2018. Towards a Framework to Manage Perceptual Uncertainty for Safe Automated Driving. In *Computer Safety, Reliability, and Security*, Barbara Gallina, Amund Skavhaug, Erwin Schoitsch, and Friedemann Bitsch (Eds.). Springer International Publishing, Cham, 439–445.

[4] Jiajun Deng, Shaoshuai Shi, Peiwei Li, Wengang Zhou, Yanyong Zhang, and Houqiang Li. 2021. Voxel r-cnn: Towards high performance voxel-based 3d object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 1201–1209.

[5] Yinpeng Dong, Caixin Kang, Jinlai Zhang, Zijian Zhu, Yikai Wang, Xiao Yang, Hang Su, Xingxing Wei, and Jun Zhu. 2023. Benchmarking Robustness of 3D Object Detection to Common Corruptions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1022–1032.

[6] Robin Heinzler et al. 2019. Weather Influence and Classification with Automotive Lidar Sensors. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. 1527–1534.

[7] Infineon Technologies AG. 2024. From ADAS to Autonomous Driving. https://www.infineon.com/cms/en/discoveries/adas-to-ad/. Accessed: 2024-02-10.

[8] Jyothish K James et al. 2018. Classification of lidar sensor contaminations with deep neural networks. In *Proceedings of the Computer Science in Cars Symposium (CSCS)*. 8.

[9] Joel Janai, Fatma Güney, Aseem Behl, Andreas Geiger, et al. 2020. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision* 12, 1–3 (2020), 1–308.

[10] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. 2018. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 751–766.

[11] Bin Lu, Yang Sun, and Zhenyu Yang. 2023. Voxel Graph Attention for 3-D Object Detection From Point Clouds. *IEEE Transactions on Instrumentation and Measurement* 72 (2023), 1–12. https://doi.org/10.1109/TIM.2023.3301907

[12] Nguyen Anh Minh Mai et al. 2022. Camera and LiDAR analysis for 3D object detection in foggy weather conditions. In *2022 12th International Conference on Pattern Recognition Systems (ICPRS)*. 1–7.

[13] MIT News. 2019. How fast can humans react to car hazards. https://news.mit.edu/2019/how-fast-humans-react-car-hazards-0807. Accessed: 2024-02-10.

[14] Birgit Schlager et al. 2022. Contaminations on Lidar Sensor Covers: Performance Degradation Including Fault Detection and Modeling as Potential Applications. *IEEE Open Journal of Intelligent Transportation Systems* 3 (2022), 738–747. https://doi.org/10.1109/ojits.2022.3214094

[15] Weijing Shi and Ragunathan (Raj) Rajkumar. 2020. Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[16] Li Tang et al. 2020. Performance Test of Autonomous Vehicle Lidar Sensors Under Different Weather Conditions. *Transportation Research Record* 2674, 1 (2020), 319–329. https://doi.org/10.1177/0361198120901681

[17] Xiaolin Tang, Kai Yang, Hong Wang, Jiahang Wu, Yechen Qin, Wenhao Yu, and Dongpu Cao. 2022. Prediction-Uncertainty-Aware Decision-Making for Autonomous Vehicles. *IEEE Transactions on Intelligent Vehicles* 7, 4 (2022), 849–862. https://doi.org/10.1109/TIV.2022.3188662

[18] Sumesh Thakur, Bivash Pandey, Jiju Peethambaran, and Dong Chen. 2022. A Graph Attention Network for Object Detection from Raw LiDAR Data. In *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*. 3091–3094. https://doi.org/10.1109/IGARSS46834.2022.9883734

[19] Jose Roberto Vargas Rivero et al. 2020. Weather Classification Using an Automotive LIDAR Sensor Based on Detections on Asphalt and Atmosphere. *Sensors* 20, 15 (2020). https://doi.org/10.3390/s20154306

[20] Hong Wang, Wenbo Shao, Chen Sun, Kai Yang, Dongpu Cao, and Jun Li. 2024. A Survey on an Emerging Safety Challenge for Autonomous Vehicles: Safety of the Intended Functionality. *Engineering* (2024). https://doi.org/10.1016/j.eng.2023.10.011

[21] Oliver Willers, Sebastian Sudholt, Shervin Raafatnia, and Stephanie Abrecht. 2020. Safety Concerns and Mitigation Approaches Regarding the Use of Deep Learning in Safety-Critical Perception Tasks. In *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops*, António Casimiro, Frank Ortmeier, Erwin Schoitsch, Friedemann Bitsch, and Pedro Ferreira (Eds.). Springer International Publishing, Cham, 336–350.

[22] Akihiro Yamaguchi, Yousuke Watanabe, Kenya Sato, Yukikazu Nakamoto, Yoshiharu Ishikawa, Shinya Honda, and Hiroaki Takada. 2017. In-vehicle distributed time-critical data stream management system for advanced driver assistance. *Journal of Information Processing* 25 (2017), 107–120.

[23] Kai Yang, Xiaolin Tang, Sen Qiu, Shufeng Jin, Zichun Wei, and Hong Wang. 2023. Towards Robust Decision-Making for Autonomous Driving on Highway. *IEEE Transactions on Vehicular Technology* 72, 9 (2023), 11251–11263. https://doi.org/10.1109/TVT.2023.3268500

[24] Yuxiao Zhang, Alexander Carballo, Hanting Yang, and Kazuya Takeda. 2023. Perception and sensing for autonomous vehicles under adverse weather conditions: A survey. *ISPRS Journal of Photogrammetry and Remote Sensing* 196 (2023), 146–177. https://doi.org/10.1016/j.isprsjprs.2022.12.021

[25] Xingyu Zhao, Kizito Salako, Lorenzo Strigini, Valentin Robu, and David Flynn. 2020. Assessing safety-critical systems from operational testing: A study on autonomous vehicles. *Information and Software Technology* 128 (2020), 106393. https://doi.org/10.1016/j.infsof.2020.106393

# Enabling Operational Data Analytics for Datacenters through Ontologies, Monitoring, and Simulation-based Prediction

Shekhar Suman [+]
s.suman@student.vu.nl
Vrije Universiteit Amsterdam
The Netherlands

Xiaoyu Chu [+]
x.chu@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

Dante Niewenhuis
d.niewenhuis@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

Sacheendra Talluri
s.talluri@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

Tiziano De Matteis
t.de.matteis@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

Alexandru Iosup
a.iosup@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

## ABSTRACT

Datacenters are key components in the ICT infrastructure supporting our digital society. Datacenter operations are hampered by operational complexity and dynamics, risking to reduce or even offset the performance, energy efficiency, and other datacenter benefits. A promising emerging technology, Operational Data Analytics (ODA), promises to collect and use monitoring data to improve datacenter operations. However, it is challenging to organize, share, and leverage the massive and heterogeneous data resulting from monitoring datacenters. Addressing this combined challenge, starting from the idea that graphs could provide a good abstraction, in this work we present our early work on designing and implementing a graph-based approach for datacenter ODA. We focus on two main components of datacenter ODA. First, we design, implement, and validate a *graph-based ontology for datacenters* that captures both high-level meta-data information and low-level metrics of operational data collected from real-world datacenters, and maps them to a graph structure for better organization and further use. Second, we design and implement *ODAbler, a software framework for datacenter ODA*, which combines ODA data with an online simulator to make predictions about current operational decisions and other what-if scenarios. We take the first steps to illustrate the practical use of ODAbler, and explore its potential to support datacenter ODA through graph-based analysis. Our work helps construct the case that graph-based ontologies have great value for datacenter ODA and, further, to improving datacenter operations.

## CCS CONCEPTS

• **Computer systems organization → Maintainability and maintenance**.

## KEYWORDS

graph-based ontology, ODAbler, OpenDC, operational data analytics, monitoring, mapping, analysis, simulation, datacenter

**ACM Reference Format:**
Shekhar Suman [+], Xiaoyu Chu [+], Dante Niewenhuis, Sacheendra Talluri, Tiziano De Matteis, and Alexandru Iosup. 2024. Enabling Operational Data Analytics for Datacenters through Ontologies, Monitoring, and Simulation-based Prediction. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3652897

## 1 INTRODUCTION

Our economy, academia, and more broadly our society rely on ICT infrastructures; for example, in the Netherlands, nearly two-thirds of the $ 1 trillion GDP, over 3 million jobs, and a large fraction of economic growth depend directly on such infrastructure [6]. Datacenters are one of the most important components of the modern ICT infrastructure. The complexity of modern datacenters data introduces significant operational challenges, including challenges related to performance, availability, and efficient use of energy. To address such challenges, we need new ways to collect, share, and understand operational data across different operational layers of datacenters, simplifying, hardware, software, and applications. In this work, we consider how to enable *Operational Data Analytics (ODA)*, a family of concepts and techniques that leverage monitoring data to extract high-level, actionable knowledge that can be used to drive operational decisions [12]. We focus in this work on how ontologies, monitoring, and simulation-based prediction can enable ODA for datacenters.

Figure 1 shows a common ODA process, derived from Netti et al. [12]. The process includes five components: (*1*) The physical *Datacenter*, which contains any kind of hardware, energy-transferring devices, and cooling infrastructure; (*2*) *Data Collection*, which includes different sources of operational data, such as live monitoring sensors, tracing frameworks, and logging, (*3*) *Data Storage*, where, in this work, we propose to augment the traditional time-series database with an ontology-based approach, (*4*) *Data Analytics*, where different techniques, such as workload characterization and modeling, are used and possibly combined, to analyze and optimize datacenter operations, and, last, (*5*) *Reassessment and Redeployment*,

[+]These two authors contributed equally to this work.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652897

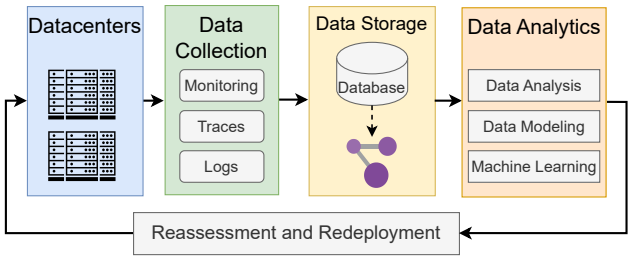



**Figure 1: ODA process.**

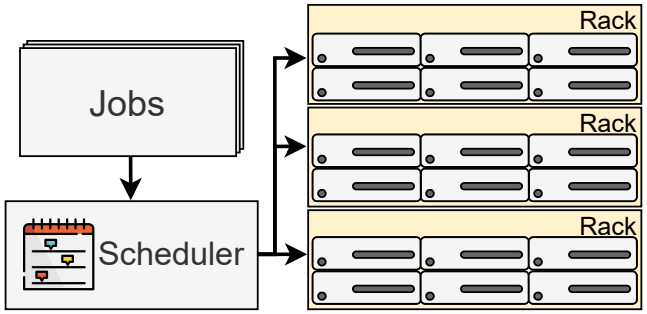



**Figure 2: System model.**

where past analytics results and current status are used to drive decisions and optimize the ODA process automatically.

We augment in this work the typical stage 3 in ODA processes with an ontology-based approach, and stage 4 with an approach that leverages the ontology and combines analytical capabilities related to it into a larger ODA framework, ODAbler. Our motivation to develop a datacenter ontology are based on the promise of graph data for any field [14]: (1) Ontology can structure and formalize complex hierarchies and relationships in graph formats and thus can be a good way to organize operational data collected from datacenters, (2) Many powerful graph-based algorithm applications can be used for ODA data, leading to insights that are difficult to obtain from non-graph data and in particular from mere time-series, and (3) From a computer systems perspective, we also aim to give insights into building graph-based datasets and data management approaches for datacenter operational data, encouraging future research in this area.

Several studies have already developed and built ontologies specifically for ICT infrastructures and datacenters. [1–3]. The CloudLightning Ontology [1] is designed to address the heterogeneous resources management interoperability issues. The HPC ontology [9] is used for managing training datasets of AI models for addressing various challenges in HPC. The ICT Infrastructures ontology network [2] is a high-level datacenter ontology that includes software, database, hardware, server, and network. However, these ontologies are neither built on the monitoring data, nor do they cover all levels and attributes of datacenters, and they are not designed for operational data analytics, which requires special functions. Thus, a data-centered ontology for operational data analytics is still lacking.

Addressing a key gap around the use of ontology-based approaches for datacenter ODA, in this work we take first steps toward a universal framework for datacenter ODA, with a twofold contribution:

(1) We design and implement an ontology to structure the data collected in datacenters (in Section 3). It is the first data-centered ontology for datacenters, which could enable complex graph analysis and applications in the future. We validate the implementation through a prototype ontology that meets the requirements for a real-world HPC cluster dataset.
(2) We design and implement the *ODAbler* framework for datacenter ODA framework (in Section 4). Based on the ontology designed for this work, ODAbler enables the ingestion and export of operational metrics typical of datacenters. It also supports complex analysis of datacenter scenarios, around a state-of-the-art simulator (here, OpenDC [11]). Last, it offers the technology framework necessary to explore, in the future, the use of graph-based analysis to understand how datacenters operate and to conduct what-if analysis for datacenters in a new way.

## 2 BACKGROUND

In this section, we describe the system model from which we collected data, and provide basic information about ontology.

### 2.1 System model

Figure 2 depicts the system model in the HPC cluster we used to collect data. A datacenter system is composed of many racks, each of which accommodates multiple server nodes. The nodes are connected through a network interconnect. Different kinds of jobs are submitted to a scheduler which then schedules them onto the nodes of the datacenter. A job can use a single node or multiple nodes.

We build the ontology based on the data collected from the *SURF Lisa cluster* by two tools. First, *SLURM* scheduler logs, from which we collect 10 months of job data, from the end of December 2021 to November 2022, with 1,596,963 records and 16 metrics. The dataset includes job-related metrics such as the number of machines allocated to jobs, the nodes that were allocated, and the completion status. Second, *Prometheus* monitor, from which we collect 5 months of node data, from June 2022 to November 2022, with 127,827,719 records and 82 metrics. This dataset includes node-related metrics such as node capacities, CPU load, memory, network, and power and temperature metrics. All these metrics are supposed to be covered in the designed ontology.

### 2.2 Ontology and OWL basics

*Ontology* is the field of science that helps us investigate what types of *entities* (or *classes*, sometimes also called *concepts*) exist in a domain of discourse, how they are grouped into categories, and how they are related to one another on the most fundamental level. An ontology along with a set of individual *instances* of classes constitutes a *knowledge base* [13]. The common reasons for developing an ontology are below [13]: (1) Sharing a common understanding of the structure of information among the utilizing resources; (2) Enabling reusability of domain knowledge; (3) Making explicit domain assumptions; (4) Analysing domain knowledge.

The *Web Ontology Language (OWL)* is a formal language for expressing ontologies and is based on the *description logic (DL)*. The underlying format that is fundamental to storing a wide range of information in OWL-based ontologies is the *Resource Description Framework (RDF)*. The information stored in an RDF consists of a triple: *subject, property, and object.* For instance, *("DataCenter-XYZ", hasAmbientTemperature, "21 degrees")* states that a *DataCenter-XYZ* has an ambient temperature of *21 degrees* (which could further be explicitly related to the time of record capture using another property, or annotated with the timestamp information).

In this project, OWL has been used for modeling the ontology (where OWL support is made available in the Python platform by *Owlready2* [8]), and a few experiments (using the *SPARQL* language) have been conducted to validate that the designed ontology meets the expectations by satisfying the requirements outlined in the next section. Some of the key definitions that set up the basis of the ontology modeling using OWL are outlined below:

- *Classes*: Entities in an object-oriented world. In an ontology, an individual can belong to several classes. The relation between the two classes can be *disjoint* (two disjoint classes cannot have individuals in common), *pairwise disjoint* (any pair made up of two classes from this list are disjoint), and *partitions* (to declare "either-or" of a class).
- *Data properties*: properties whose values are data (number, text, date, boolean, etc.). The data properties have a *domain* (the class for which the property is defined), and *range* (the associated datatype, which can be an integer or a real number, boolean, character string, date, and so on).
- *Object properties*: properties whose values are entities (i.e. ontology individuals). The range of object properties is the class of associated objects.

Generally, developing an ontology includes the following practical steps [13]:

1. Determine the domain and scope of the ontology.
2. Consider reusing existing ontologies.
3. Enumerate important terms in the ontology.
4. Define the classes and the class hierarchy.
5. Define the data properties of classes.
6. Define the object properties of classes.
7. Create instances.

# 3 ONTOLOGY DESIGN

In this section, we first analyze the requirements for the proposed datacenter ontology. Following the requirements, we design and implement the ontology according to the metrics collected from an HPC datacenter, both high-level and detailed. We also explore how they could be useful for Operational Data Analytics (ODA). Finally, we conduct validations to ensure that the ontology aligns with the proposed requirements.

Following the ontological process introduced in Section 2.2, we determine the domain and scope of the ontology through requirements analysis in Section 3.1 as step 1. We reuse the existing ontology in Section 3.2 as step 2. We do high-level design including identifying important terms in Section 3.3 as step 3. We cover steps 4-6 in in Section 3.4, and create instances for validation in Section 3.5.

**Table 1: The overall statistics of the designed ontology, based on metrics from the SURF Lisa dataset.**

| | |
|---|---|
| Axiom count | 710 |
| Logical axiom count | 442 |
| Declaration axiom count | 230 |
| Class count | 82 |
| Object property count | 17 |
| Data property count | 63 |
| Individual count count | 69 |
| Annotation property count | 2 |

## 3.1 Requirements analysis

In order to support operational data analytics, the datacenter ontology has to cover a large scope, including infrastructure, hardware, software, and applications. Besides, we give four functional requirements and two non-functional requirements for the datacenter ontology.

### 3.1.1 *Functional requirements.*

**FR1. Time-series modeling.** The ontology should support the modeling of attributes extracted from time-series metrics collected from a large-scale computing infrastructure, e.g., an HPC cluster.

**FR2. Resource description.** The ontology should describe the datacenter cluster resources in a structured way, including details about nodes, processors, etc.

**FR3. Performance metrics.** The ontology should capture and analyze performance metrics such as resource utilization and energy consumption.

**FR4. Consistency and accuracy.** The ontology should be consistent and accurate in its representation by reflecting the state of the datacenter and its resources.

### 3.1.2 *Non-functional requirements.*

**NFR1. Interoperability.** The ontology should be designed with interoperability in mind, which should facilitate integration or reuse with/by other ontologies.

**NFR2. Usability.** The ontology should be user-friendly, having sufficient comments or labels for accessibility by both experts and non-experts.

## 3.2 Reusing existing ontologies

The best resource that is closer to our ontology requirements is a work of literature studying the ontological representation of time-series observations on the Semantic Sensor Web [4]. It suggests the usage of three important modeling classes, out of which we find that the most relevant class *"Observation"* can be reused, and thus discussed in detail below. The two other classes *"Observation-Collection"* and *"TimeSeriesObservation"* do not seem convincingly reusable, mainly because of the nature of the requirements to model a sample time-series data. If we were to model multiple time-series data in the ontology, then we could have inherited the same structure. But, in this project, we limit ourselves to demonstrating a single set of records in the ontology model, other than the established relations amongst the classes. Some of the relationships for observations that have been used here are listed below:

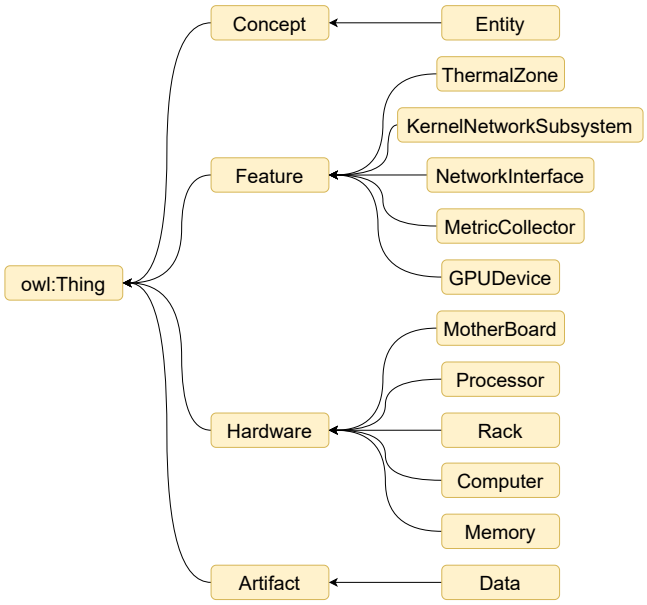



**Figure 3: High-level classes of the datacenter ontology. The arrow denote the "is-a" relation between classes.**

- *featureOfInterest*: Representation of the object being observed.
- *observedProperty*: The phenomenon for which the observation result provides an estimate of its value.
- *samplingTime* or *generatedAtTime*: The time when the phenomenon was measured.
- *result* or *value*: An estimate of the value of a property generated using a known procedure.
- *memberOf*: A relation to a set of observations of observation collection.

## 3.3 High-level design and classes

The ontology has been developed to structure and formalize complex hierarchies and relationships of operational data in datacenters, it provides a foundation to enable further graph-based applications. We first give an overview of the statistics of the designed ontology, then we describe the high-level classes, and the key subclasses step by step.

The ontology has been developed in the OWL language by using *Protégé*[1] as the ontology editor, knowledge management, and visualization system. We follow a common naming convention when defining the ontology terms: Singular nouns in CamelCase are used to present a Class, while Property names start with lowercase letters. The statistic of the implemented prototype is shown in Table 1. The ontology has 82 classes, 17 object properties, and 63 data properties in total.

*3.3.1 High-level classes.* Figure 3 depicts the high-level classes in the designed ontology. It consists of: a *Concept* class, which describes the *Entity* in datacenter, such as *Software* entity, which includes *JobScheduler, MetricsExporter, ResourceManager, MonitoringSystem* and so on; a *Feature* class describing different metric collector in the datacenter; a *Hardware* class, which defining hardware configurations such as *Processor, Memory*; a *Artifact* class, which captures various source of *Data* from *Job* or *MonitoringMetrics*.

[1]Protege - https://protege.stanford.edu/

**Table 2: Object properties of the datacenter ontology.**

| Property | Domain | Range |
|---|---|---|
| atLocation | Thing | Thing |
| exportMonitoringMetrics | MetricsExporter | MonitoringMetrics |
| featureOfInterest | Thing | Feature |
| hasExitCode | Thing | ExitCode |
| hasJobScheduler | Computer | JobScheduler |
| hasMember | Thing | Thing |
| hasMetricsExporter | Entity | MetricsExporter |
| hasMonitoringSystem | Computer | MonitoringSystem |
| hasResourceManager | Computer | ResourceManager |
| isScheduledOn | Data | Thing |
| isScheduledOnServer | Thing | Server |
| manageJob | Software | Job |
| managesJob | JobScheduler | Job |
| measuresValueOfThing | MonitoringMetrics | Thing |
| hasMember | Thing | Thing |
| observedProperty | Thing | Property |

*3.3.2 Key subclasses.* *MonitoringMetrics* is an important subclass of class *Data*, where we map the metrics collected in the dataset to the ontology. Here we capture different kinds of metrics of datacenter including energy-related data such as *Temperature, EnergyUsage*, and resource-related data such as *CPULoadAverage, NumberOfIOs*. *MetricCollector* is a subclass of class *Feature*, which shows different data collectors such as *CPULoadCollector, MemoryStatisticsCollector*. The subclasses of class *Hardware* shows the common components of a datacenter, including *Server, Rack, Processor* etc.

## 3.4 Detailed design and propertites

There are two kinds of properties: *object properties* are whose values are entities (i.e., ontology individuals), *data properties* are whose values are data (numbers, texts, dates, Booleans, etc.). In this subsection, we will introduce the details of object and data properties in our ontology.

*3.4.1 Object Properties.* Object properties indicate the relationships between two classes. Table 2 shows the information of object properties in the designed ontology. The class *Thing* has eight relationships with other classes, including: *atLocation*, presents *Thing* is at some location; *featureOfInterest*, which describes the feature being observed; *hasExitCode*; *hasMember*, indicates the membership relation between two entities, and *memberOf* is an inverse relation of it; *isScheduledOnServer*, indicates job is scheduled on some server (node); *observedProperty*, which provides an estimation of observed value. The class *Computer* has three relations: *hasJobScheduler, hasMonitoringSystem, hasResourceManager*, which reveals the relations between hardware and software. The object properties show the complexity of hierarchies in different layers inside a datacenter, so it is the key to linking different components and metrics.

*3.4.2 Data Properties.* Figure 4 shows partial data properties in this ontology. Since the goal of ontology is to better organize the operational data generated in datacenter, it should cover time-series metrics as required. The data properties include all the metrics we can collect in the SURF Lisa dataset, and the range of these properties is either float or string.

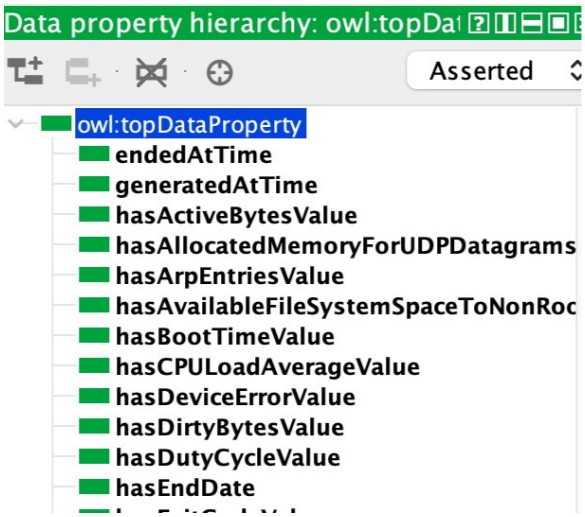



**Figure 4: Particial data properties of the datacenter ontology.**

## 3.5 Validation

This section is to validate that the modeled ontologies meet the requirements of an ontology in terms of structural modeling. It should be noted that the overall ontology structure could be validated to be syntactically correct using the graph dump property (*ontology.graph.dump* in Python's *Owlready2* module[2]), to ensure that there are no errors encountered while representing data with the ontology (e.g., datatype mismatch, incorrect attribute-name, data missing scenario). If the modeled ontology is structurally incorrect while adding data, then the graph dump statement should give an error.

*3.5.1 Validation of key properties.* Validation of some of the key properties that are common in both ontologies (e.g. ambient temperature, host power usage, etc.). We perform a SPARQL query to validate that the results are matched in both ontologies, as shown in the listing below:

```
result_surf = list(default_world.sparql("""
    PREFIX  <https://example.org/hpcontology_surf.owl#>
    SELECT DISTINCT ?x where{
    ?x rdfs:subClassOf* Property .
    }
    """))
```

*3.5.2 Validation of graph structure.* Verification that the modeled ontology also reflects the graphical layout, which could be inferred to create general graph structures on the fly for analytical purposes (e.g., for performing ODA-related analysis). The goal of this experiment is to verify that the ontology modeled for both HPC clusters can be visualized in the form of graphical layouts with nodes and edges, which could be easily converted to a graphical structure in a graph database using available tools. The resulting graphical layout for SURF's LISA cluster is visualized using *WebVOWL* ontology visualization and shown in Figure 5, from which we can see properties such as *NodeTemperature, PowerUsage, FileSystemSize* are presented in a graph format, with the relations to other metrics in the ontology.

[2]Owlready2 - https://owlready2.readthedocs.io/en/latest/

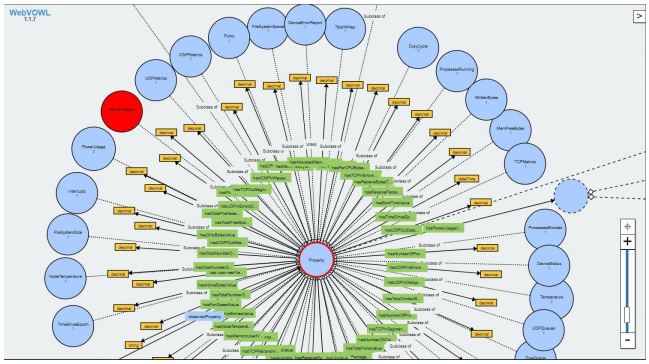


**Figure 5: SURF's OWL ontology visualization (using WebVOWL) showing the graph layout according to the VOWL specification, and listing graphical nodes and edges information on the right side information box.**

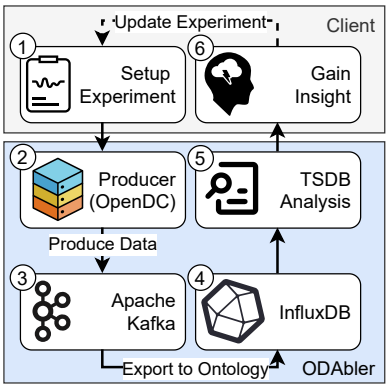



**Figure 6: High-level architecture of ODAbler framework.**

# 4 ODABLER: DESIGN OF ONTOLOGY-BASED SIMULATION IN OPENDC

We propose and implement ODAbler as a prototype to show how the ontology can benefit operational data analytic.

## 4.1 Architecture of ODAbler framework

The key design element of ODAbler involves several key elements as represented in Fig. 6. Some of the key elements include OpenDC itself, and *Apache Kafka* as the middleware which is responsible for message publishing to a time-series database *InfluxDB*, using the *Telegraf* agent. Once the time-series data is available at *InfluxDB*, the ODAbler client application performs out-of-band analysis (as part of the current scope of its implementation)

## 4.2 Implementation of the ODAbler framework

*4.2.1 Ontology-driven export of OpenDC metrics.* As shown in Table 3, the corresponding attributes or properties are selected to be exported to the time-series database *InfluXDB* for further analysis by the ODAbler analysis client. This screenshot is taken from the OpenDC's *MonitoringMetrics* class, which is a data class whose sole purpose is to hold data of various metrics as reflected by the name of the member properties.

**Table 3: Metrics exported to time-series database reusing the concepts from the designed ontology in the section 3.**

| Name | Type |
|---|---|
| cpuIdleTime | Long |
| cpuActiveTime | Long |
| cpuLostTime | Long |
| energyUsage | Double |
| upTime | Long |
| serverId | Int |
| timestamp | Long |
| cpuUtilization | Double |
| powerUsage | Double |
| guestsRunning | Int |
| policyId | Int |

*4.2.2 Technical implementation.* The technical implementation of the ODAbler framework involves the following steps: (1) Enable fault injection in OpenDC; (2) Launch InfluxDB and Kafka; (3) Start Telegraf service; (4) Launch OpenDC server, then launch ODAbler client analyzer application. (5) Export operational data from OpenDC to Apache Kafka. (6) Kafka exports the ontology-driven relevant power usage and energy usage metrics (besides others) to InfluxDB via the Telegraf agent. (7) ODAbler performs out-of-band analysis on InfluxDB data, once the data is fully available.

## 4.3 Exploration of graph applications

We explore the potential applications of ontology-based graph applications for datacenter operational data analytics. (We have not yet built these capabilities in ODAbler, but plan to do so.)

*4.3.1 Graph queries.* Application performance and behavior are linked across the hardware-software stack. However, the metrics are isolated in different parts of the stack. Ontology-enabled monitoring and analysis tools help link metrics across the stack. Listing 1 provides an example query to access all machines running an application using the "production" database and the p99 latency of the connection between the application and the database.

```
SELECT ?appname, ?machine, p99(?latency)
WHERE
  {
    ?x    appname       ?appname ;
          isScheduledOn ?machine ;
          linkedTo      ?link .
    ?link linkedTo      ?db .
          hasMetric     ?y .
    ?y    metricname    "latency" .
          value         ?latency ;
    ?db   appname       "production" .
  }
```

**Listing 1: Example query to retrieve p99 latency of all apps connected to the "production" database.**

*4.3.2 Graph analysis.* The typical data center data analysis workflow now involves manually collating and analyzing metrics data. New databases [7] have demonstrated the benefit of graph-aware query engines for linked data. However, the link information is unique to each data center, hindering the development of ODA-specific databases and limiting us to slow ad-hoc data analysis. A common ontology would allow data center operators to bring powerful tools to bear on operational data analytics [12] and workload modeling [15].

*4.3.3 Graph machine learning.* Machine learning has proven promising in data center resource management applications [10, 15]. However, automatically enhancing datacenter processes remains a challenge. Data availability and domain shift are two obstacles to pervasive machine learning in the data center. Each datacenter has its own idiosyncratic data collection architecture, and ML systems trained on one datacenter's data do not prove helpful in other datacenters. Data normalized using a common ontology helps tackle these obstacles.

# 5 RELATED WORK

An ontology encompasses a representation, formal naming, and definition of the categories, properties, and relations between the concepts, data, and entities that substantiate one, many, or all domains of discourse. The basic idea is to represent the properties of a subject area and their relationships, by defining a set of concepts and categories that represent the subject. One of the main reasons for designing an ontology for HPC is to make training datasets and AI models FAIR (FAIR data principles describe Findability, Accessibility, Interoperability, and Reusability) [9]. Some of the existing HPC ontology design already captures both high-level meta information as well as low-level data content for software, hardware, experiments, workflows, training datasets, AI models, and so on [3].

HPC ontology modelling work has already been done in previous scientific research. There are several research works already done in the field of HPC ontology, for example, by C. Liao et al. [9], and by Castañé et al. [1], amongst others. One of the comprehensive HPC ontology designs that already captures both high-level meta information as well as low-level data content for software, hardware, experiments, workflows, training datasets, AI models, and so on is available as HPC Ontology. There are other works of literature on HPC resources' ontology models like Zhou et al. [19], Zhao et al. [18], Tenschert [16] and others, but those are simplified where the main goal of the authors has been to decompose applications between compute and data processes for HPC environments. There are several other works of literature presenting a unified ontology of cloud computing like Youseff et al. [17] and Imam [5]. Amongst these works, we did not find any literature studying the modelling of ontology derived from the metrics collected from operational HPC clusters. But, these metrics are the source of ODA framework design, and, so, the ODA framework should be driven by the modelled ontology, which should be derived from those captured metrics of large-scale computing infrastructures.

# 6 CONCLUSION AND FUTURE WORK

This paper presents our ongoing efforts to build a datacenter ontology to enable operational data analytics based on the data from a real-world HPC cluster. We adopt both high-level and detailed designs to cover the designed requirements. The resulting datacenter ontology has modeled properties of essential concepts of this domain, including hardware, software, and collected metrics.

[3] https://hpc-fair.github.io/ontology/

Through the validation, the ontology can support typical search queries using SPARQL.

An essential item of future work is to incorporate graph-based analytics into ODAbler, and, further, explore applications of this technology. Although the exact capabilities of a graph-based ODAbler, and more generally of graph-based datacenter ODA, are largely unknown, this line of future work will bring evidence of whether our exploration in Section 4.3 is correct and could result in a new way of understanding datacenters.

Future work will also include extensions of the current ontology, such as more comprehensive relations between different entities. We will also add more individuals to the ontology and conduct graph-based experiments to see if the ontology can help better understand the datacenter operation. The current draft ontology is available online at *https://github.com/am-i-helpful/hpc-ontology-modeller*.

## ACKNOWLEDGEMENT

We thank the Dutch National Supercomputing Center SURF for providing the data. We thank the China Scholarship Council (CSC) for supporting Xiaoyu Chu. We thank the support of Netherlands-funded projects NWO OffSense and GFP 6G FNS, and EU-funded projects MCSA-RISE Cloudstars and Horizon Graph-Massivizer.

## REFERENCES

[1] Gabriel G. Castañé, Huanhuan Xiong, Dapeng Dong, and John P. Morrison. 2018. An ontology for heterogeneous resources management interoperability and HPC in the cloud. *Future Gener. Comput. Syst.* 88 (2018), 373–384. https://doi.org/10.1016/j.future.2018.05.086

[2] Óscar Corcho, David Chaves-Fraga, Jhon Toledo, Julián Arenas-Guerrero, Carlos Badenes-Olmedo, Mingxue Wang, Hu Peng, Nicholas Burrett, Jose Mora, and Puchao Zhang. 2021. A High-Level Ontology Network for ICT Infrastructures. In *The Semantic Web - ISWC 2021 - 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24-28, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12922)*, Andreas Hotho, Eva Blomqvist, Stefan Dietze, Achille Fokoue, Ying Ding, Payam M. Barnaghi, Armin Haller, Mauro Dragoni, and Harith Alani (Eds.). Springer, 446–462. https://doi.org/10.1007/978-3-030-88361-4_26

[3] Yu Deng, Ronnie Sarkar, HariGovind V. Ramasamy, Rafah Hosn, and Ruchi Mahindru. 2013. An Ontology-Based Framework for Model-Driven Analysis of Situations in Data Centers. In *2013 IEEE International Conference on Services Computing, Santa Clara, CA, USA, June 28 - July 3, 2013*. IEEE Computer Society, 288–295. https://doi.org/10.1109/SCC.2013.98

[4] Cory Andrew Henson, Holger Neuhaus, Amit P Sheth, Krishnaprasad Thirunarayan, and Rajkumar Buyya. 2009. An ontological representation of time series observations on the semantic sensor web. (2009).

[5] Fahim T Imam. 2016. Application of ontologies in cloud computing: The state-of-the-art. *arXiv preprint arXiv:1610.02333* (2016).

[6] Alexandru Iosup, Fernando Kuipers, Ana Lucia Varbanescu, Paola Grosso, Animesh Trivedi, Jan S. Rellermeyer, Lin Wang, Alexandru Uta, and Francesco Regazzoni. 2022. Future Computer Systems and Networking Research in the Netherlands: A Manifesto. *CoRR* abs/2206.03259 (2022). https://doi.org/10.48550/ARXIV.2206.03259 arXiv:2206.03259

[7] Guodong Jin, Xiyang Feng, Ziyi Chen, Chang Liu, and Semih Salihoglu. 2023. KÙZU Graph Database Management System. In *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023*. www.cidrdb.org. https://www.cidrdb.org/cidr2023/papers/p48-jin.pdf

[8] Jean-Baptiste Lamy. 2017. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial intelligence in medicine* 80 (2017), 11–28.

[9] Chunhua Liao, Pei-Hung Lin, Gaurav Verma, Tristan Vanderbruggen, Murali Emani, Zifan Nan, and Xipeng Shen. 2021. HPC Ontology: Towards a Unified Ontology for Managing Training Datasets and AI Models for High-Performance Computing. In *IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments, MLHPC@SC 2021, St. Louis, MO, USA, November 15, 2021*. IEEE, 69–80. https://doi.org/10.1109/MLHPC54614.2021.00012

[10] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*, Jianping Wu and Wendy Hall (Eds.). ACM, 270–288. https://doi.org/10.1145/3341302.3342080

[11] Fabian Mastenbroek, Georgios Andreadis, Soufiane Jounaid, Wenchen Lai, Jacob Burley, Jaro Bosch, Erwin Van Eyk, Laurens Versluis, Vincent Van Beek, and Alexandru Iosup. 2021. OpenDC 2.0: Convenient modeling and simulation of emerging technologies in cloud datacenters. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 455–464.

[12] Alessio Netti. 2022. *Holistic and Portable Operational Data Analytics on Production HPC Systems*. Ph. D. Dissertation. Technische Universität München.

[13] Natalya F Noy, Deborah L McGuinness, et al. 2001. Ontology development 101: A guide to creating your first ontology.

[14] Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid G. Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, and Eiko Yoneki. 2021. The future is big graphs: a community view on graph processing systems. *Commun. ACM* 64, 9 (2021), 62–71. https://doi.org/10.1145/3434642

[15] Siddharth Samsi, Matthew L. Weiss, David Bestor, Baolin Li, Michael Jones, Albert Reuther, Daniel Edelman, William Arcand, Chansup Byun, John Holodnack, Matthew Hubbell, Jeremy Kepner, Anna Klein, Joseph McDonald, Adam Michaleas, Peter Michaleas, Lauren Milechin, Julia S. Mullen, Charles Yee, Benjamin Price, Andrew Prout, Antonio Rosa, Allan Vanterpool, Lindsey McEvoy, Anson Cheng, Devesh Tiwari, and Vijay Gadepally. 2021. The MIT Supercloud Dataset. In *2021 IEEE High Performance Extreme Computing Conference, HPEC 2021, Waltham, MA, USA, September 20-24, 2021*. IEEE, 1–8. https://doi.org/10.1109/HPEC49654.2021.9622850

[16] Axel Tenschert. 2016. Ontology matching in a distributed environment. (2016).

[17] Lamia Youseff, Maria Butrico, and Dilma Da Silva. 2008. Toward a unified ontology of cloud computing. In *2008 Grid Computing Environments Workshop*. IEEE, 1–10.

[18] Y Zhao, C Liao, and X Shen. 2017. *An Infrastructure for HPC Knowledge Sharing and Reuse*. Technical Report. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

[19] Aolong Zhou, Kaijun Ren, Xiaoyong Li, Wen Zhang, and Xiaoli Ren. 2019. Building quick resource index list using wordnet and high-performance computing resource ontology towards efficient resource discovery. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 885–892.

# ExaQuery: Proving Data Structure to Unstructured Telemetry Data in Large-Scale HPC

Junaid Ahmed Khan
DEI Department, University of Bologna
Italy
junaidahmed.khan@unibo.it

Martin Molan
DEI Department, University of Bologna
Italy
martin.molan2@unibo.it

Matteo Angelinelli
HPC Department, CINECA
Italy
m.angelinelli@cineca.it

Andrea Bartolini
DEI Department, University of Bologna
Italy
a.bartolini@unibo.it

## ABSTRACT

High-performance computing (HPC) is the cornerstone of technological advancements in our digital age, but its management is becoming increasingly challenging, particularly as systems approach exascale. Operational data analytics (ODA) and holistic monitoring frameworks aim to alleviate this burden by collecting live telemetry from HPC systems. ODA frameworks rely on NoSQL databases for scalability, with implicit data structures embedded in metric names, necessitating domain knowledge for navigating telemetry data relations. To address the imperative need for explicit representation of relations in telemetry data, we propose a novel ontology for ODA, which we apply to a real HPC installation. The proposed ontology captures relationships between topological components and links hardware components(compute nodes, rack, systems) with job's execution and allocations collected telemetry. This ontology forms the basis for constructing a knowledge graph, enabling graph queries for ODA. Moreover, we propose a comparative analysis of the complexity (expressed in lines of code) and domain knowledge requirement (qualitatively assessed by informed end-users) of complex query implementation with the proposed method and NoSQL methods commonly employed in today's ODAs. We focused on six queries informed by facility managers' daily operations, aiming to benefit not only facility managers but also system administrators and user support. Our comparative analysis demonstrates that the proposed ontology facilitates the implementation of complex queries with significantly fewer lines of code and domain knowledge required as compared to NoSQL methods.

## CCS CONCEPTS

• **Information systems → Resource Description Framework (RDF)**; • **Computing methodologies → Ontology engineering**.

## KEYWORDS

High performance computing (HPC), Operational Data Analytics(ODA), Resource Description Framework (RDF) ontology, SPARQL

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652898

**ACM Reference Format:**
Junaid Ahmed Khan, Martin Molan, Matteo Angelinelli, and Andrea Bartolini. 2024. ExaQuery: Proving Data Structure to Unstructured Telemetry Data in Large-Scale HPC. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3652898

## 1 INTRODUCTION

The rise in complexity of large-scale computing infrastructures driven by post Moore's and Dennard's scaling era presents unprecedented challenges. Key challenges include efficient power management, optimization for parallelism, data movement and storage, software complexity, fault tolerance, scalability, workload diversity, resource allocation, and security. Many data centers explore Operational Data Analytics (ODA) to extract knowledge from monitoring data, enabling control over system parameters and aiding administrators through visualization. Despite extensive research into individual aspects of ODA, comprehensive solutions for production remain rare, particularly given the inherent complexity of HPC[9, 13].

HPC is operated by multiple teams and organizations, each tasked with distinct responsibilities for production. This includes system administrators, facility managers, and user support, who collectively contribute to its efficient operation and management. ODA targets holistic management, where the data includes diverse types such as job tables, sensor time-series data, and other varied representations ranging from log files and configuration files to system metadata. ODA frameworks often rely on NoSQL databases as they allow flexibility with diverse data sources and scalability to handle big data frameworks[11]. Moreover, namespaces adopted in ODA are tailored to the specifics of vendors, sites, or configurations, jeopardizing the portability of knowledge extraction solutions.

Acquiring domain knowledge presents a formidable challenge, as it often relies on undisclosed or dispersed information within various organizations and teams managing similar resources, leading to a fragmented understanding. In ODA, data demonstrates interconnectivity and the true value lies in identifying and harnessing these complex relationships. These relationships encompass various aspects, including the interactions between system components, submitted jobs, their execution on specific compute nodes, event correlations, and topology mapping.

In this work, we propose the first ontology aiming to provide a structured data model that captures these intricate relationships. The current state-of-the-art data center ontologies focus on inventory and infrastructure[4, 5], while the proposed ontology goes further by incorporating topological component relationships and establishing links between hardware components (such as compute nodes, racks, and systems) and job data. This ontology serves as the foundation for constructing a knowledge graph, providing a structured representation of ODA data, facilitating organized retrieval of interconnected data using graph queries. This ontology has been developed specifically for the CINECA Italian Tier-0 supercomputing center[15]. We utilized the Marconi100 (M100) system at CINECA, which employs the Examon ODA framework for holistic monitoring (detailed in sec.3), operating on Cassandra DB and KairosDB (a NoSQL time-series database), utilizing an encoded version of metric names and properties as column names. The results of this manuscript were obtained using a subset of publicly available M100 Examon collected data [3]

Furthermore, this manuscript includes a comparative analysis of query implementation complexity, measured in lines of code (LOC), and domain knowledge required between ontology-based approaches and NoSQL methods. A lower LOC indicates simpler code, while qualitative assessment of domain knowledge requirements is pivotal in determining the user-friendliness of the proposed ontology. The objective is to underscore the significance of ontologies for ODA and illustrate how they can facilitate ODA for HPC.

## 2 RELATED WORK

In this manuscript, we target the development of ontologies for data centers and HPC suitable for ODAs telemetry. With this regard, Oscar Corcho et al.[5] identify a lack of comprehensive implementations and common data models not only in this field but also across other ICT infrastructure areas. Their work is deemed impactful, showcasing the practical use of ontologies in managing data heterogeneity. Gabriel G. Castañé et al.[4], propose an ontology integrating HPC and cloud. However, its emphasis on HPC-cloud interrelations may limit its relevance to our specific requirement of simplifying query of telemetry data in HPC. Liao et al.[7] introduce an HPC ontology to ensure FAIRness (Findable, Accessible, Interoperable, Reusable) of training datasets and AI models on heterogeneous supercomputers. Their ontology offers controlled vocabularies and formal knowledge representations for data annotation and SPARQL query support, which is not the target of the proposed manuscript. Kousha et al.[6] focus on an HPC ontology tailored for job script submission and AI-assisted tools, unlike this paper which concentrates on ODA telemetry data retrieval. Additionally, Tuovinen et al.[14] present an HPC ontology to make a unified framework capable of adapting queries across different time-series storages. In contrast, the ontology proposed in this manuscript is designed to address a specific set of queries essential for the daily operations of an HPC facility manager/engineer. The aim is to simplify query implementations and reduce the required domain knowledge compared to NoSQL approaches. Additionally, we validate our approach through a comparative analysis to demonstrate its simplicity, thus proving the adoption of data structures to handle unstructured telemetry data in large-scale HPC.

## 3 BACKGROUND: EXAMON

Examon is a holistic monitoring framework for HPC[2]. It is designed to collect data from various sources, including hardware sensors, software logs, and performance metrics, and stores this data in a NoSQL database (Cassandra, with KairosDB for time-series) in a centralized repository.

Examon's data collection targets a diverse range of sources, as depicted in (Fig.1). The complexity of the collected data encompasses hardware sensors—such as CPU load across all cores, CPU clock, instructions per second, memory accesses, power consumption, fan speed, and ambient and component temperatures—along with workload-related information like job submissions and their characteristics. Additionally, Examon actively monitors compute node availability by capturing warning messages and alarms from diagnostic software tools used by system administrators. The figure further illustrates the granularity of Examon's approach, showcasing separate plugins for each hardware component, each equipped with specific sensors. This design underscores Examon's capacity to manage diverse data sources, contributing to its inherent capability to handle massive data complexity in monitoring. The openly available dataset by Borghesi et al.[3] covers a spectrum of metrics, from hardware parameters to system-related statistics.

Furthermore, Examon employs a specific set of parameters and tags, and to interact with its dataset, it features a dedicated query language known as ExamonQL. This language allows users to access information stored in the database, including metadata, and generate dataframes of the queried results.

## 4 METHODOLOGY

The methodology involves creating a knowledge graph aligned with Examon's operational principles. This section details the proposed ontology, its specifications, query language for ontology, complex queries for comparison with ExamonQL, and the evaluation criteria for the comparative analysis.

### 4.1 ODA ontology

In this subsection, we outline the reasoning behind the proposed ontology, followed by its explanation. The Resource Description Framework (RDF) plays a central role in this context, being a web standard essential for ontologies and knowledge graphs. Employing a triple structure—comprising subject, predicate, and object—RDF efficiently represents relationships. In RDF, the Uniform Resource Identifier(URI) uniquely identifies resources, such as classes and properties. These URIs can be in the form of Uniform Resource Locator(URL), providing the means to locate a resource on the internet. In the context of the proposed paper, the resources refer to components and telemetry data. RDF's flexibility in accommodating both literal values and resource descriptions makes it an invaluable tool for constructing ontologies, providing structured models to define concepts and their relationships[8].

*4.1.1 Reasoning behind ontology.* The proposed ontology follows a novel approach that exploits the holistic nature of ODA's (and

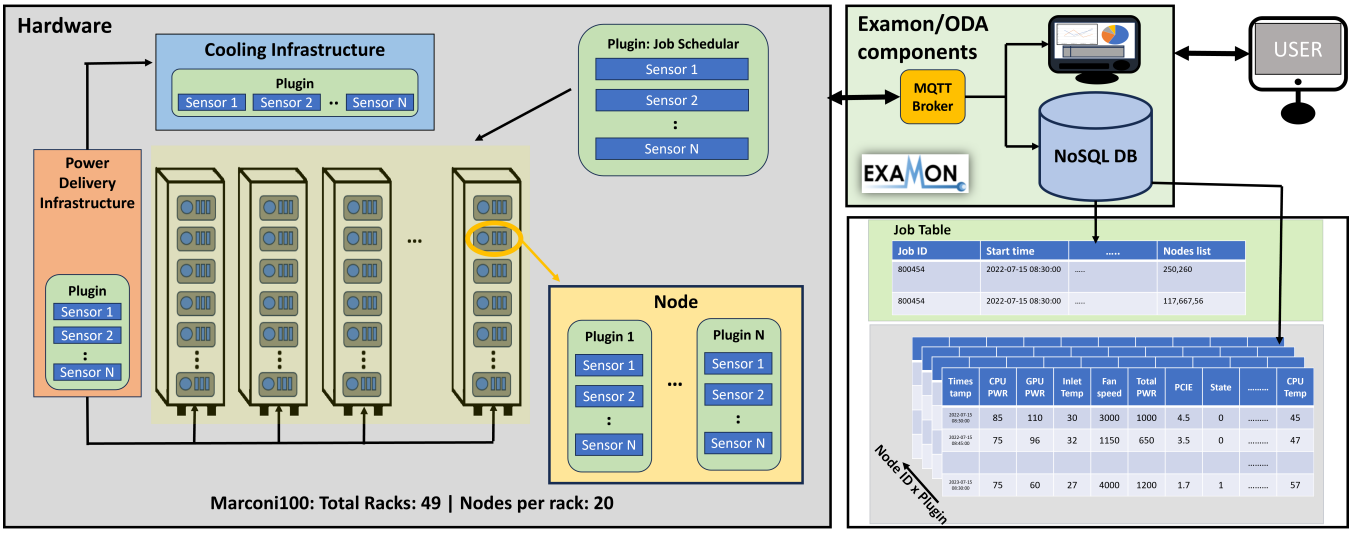



**Figure 1: Examon's massive scale and data heterogeneity.**

Examon's) monitoring data and the natural ability of knowledge graphs to capture relationships between data. As this ontology is designed to facilitate the work of large-scale HPC center data analysts and facility managers, it is designed to best meet the needs of these users. While Examon is a very powerful tool for holistic monitoring, it requires a thorough knowledge of the data architecture itself. With the proposed ontology, data is organized in a structure that allows easy interrogation by end users. In particular, as will be shown in the following sections, the data analysis process is greatly simplified, allowing a data-driven usage, management, and optimization of supercomputer systems production with workloads such as those proposed by Molan et al.[10].

*4.1.2 Ontology creation process.* The proposed ontology is developed to establish logical connections among the various data sources within Marconi100, as perceived by system administrators such as facility engineers and managers. Aligned with the underlying principles of Examon (see sec.3), it caters to the meticulous organization of telemetry data illustrated in Figure 1. In Examon, telemetry data is structured in a Plugin-centric manner, with specific plugins housing sensors tailored to each resource within the facility, be it a compute node or a component of the cooling infrastructure. These sensors gather data, which is then stored in individual files within their respective folders in the database, following a clear pathway from Plugin to Sensor to Sensor Reading, culminating in a storage file termed as a "Data Record" within our proposed ontology (see Fig.2).

However, Examon lacks inherent topological information crucial for understanding the physical organization and location of resources, particularly significant for workloads involving graph processing[10]. In an HPC facility, the natural topological structure typically revolves around compute nodes housed in racks, each rack assigned a physical location in the x and y dimensions, with compute nodes stacked within. Consequently, the position of a compute node within the stack becomes the third dimension, denoted as "Position" in our proposed ontology.

Moreover, an integral aspect of any HPC system is the jobs submitted to it. Therefore, our proposed ontology incorporates job-specific information, establishing a natural linkage between submitted jobs and the resources they utilize, which are compute nodes. This holistic approach creates a unified framework wherein every resource within the HPC facility is interconnected with its logical connections—an aspect lacking in the monitoring framework of Examon.

*4.1.3 Proposed Ontology.* The proposed ontology (Fig.2) presents a significant improvement for ODA in HPC. This structured framework organizes elements such as racks, nodes, positional information, plugin-specific sensors, and their readings. It establishes explicit relationships between HPC and ODA components, including a specific link between submitted job and the resources utilized, a feature lacking in other approaches[4, 5]. The proposed ontology provides a comprehensive model for integrating and understanding sensor data, spatial configurations, job execution, and deployed software/hardware components status in HPC infrastructure.

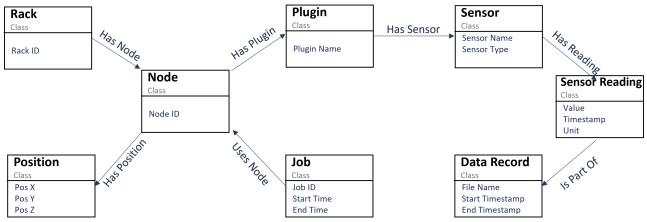



**Figure 2: Proposed Ontology**

HPC cluster topology consists of multiple racks, each housing a set of compute nodes. ODA frameworks[11] also organize data into different plugins (9 in Examon: Nagios, Ganglia, IPMI, Job table, Slurm, etc), each linked to its corresponding monitored sensors. The proposed ontology mirrors these observations by representing physical components as classes and capturing associated information through properties. Relationships between classes are precisely

defined, aligning with the arrangement of plugins and sensors of Marconi100 and Examon.

Table 1 provides an overview of the proposed ontology's classes and their attributes, where each class represents a component within the HPC system. Table 2 reports the properties of the proposed ontology, outlining their roles and functionalities, which establish relationships between classes.

| Class Name | Description |
|---|---|
| Sensor | Represents individual sensors with attributes: `sensorName` and `sensorType`. |
| Sensor Reading | Captures sensor data with attributes: `value`, `timestamp`, and `unit`. |
| Plugin | Represents specific plugins, identified by `pluginName`. |
| Job | Represents job information with attributes: `jobId`, `startTime`, and `endTime`. |
| Rack | Represents physical racks with a unique `rackId`, that houses nodes. |
| Node | Represents individual compute nodes with a unique `nodeId` |
| Position | Defines spatial coordinates (`posX`, `posY`, `posZ`) of nodes. |
| Data Record | Represents stored data records in database with attributes: `fileName`, `startTimestamp`, and `endTimestamp`. |

**Table 1: Classes Overview**

| Property | Description |
|---|---|
| Has Plugin | Connects nodes to its plugins. |
| Has Sensor | Links plugins to associated sensors in the HPC infrastructure. |
| Has Reading | Connects sensors to its readings. |
| Uses Node | Associates jobs with the nodes utilized during the job's execution. |
| Has Node | Establishes a relationship between racks and nodes. |
| Has Position | Connects nodes to spatial coordinates (X, Y, Z) representing their physical location. |
| Is Part Of | Links sensor readings to data record, specifying their inclusion in specific files in storage. |

**Table 2: Properties**

## 4.2 Knowledge graph: Ontology realisation

Ontology is a structured way of representing knowledge, defining concepts and relationships. Meanwhile, a knowledge graph is a graph-based structure built upon the schema set by the ontology, representing information in nodes(components) and edges(relations between components). By constructing a knowledge graph based on the proposed ontology, we enable the implementation of graph queries. These queries would be utilized for the comparative analysis between NoSQL methods. The evaluation criteria are explained in the (sec.4.5).

## 4.3 ODA Complex Queries

Table 3 reports the complex ODA queries. Query 1,2,3 targets anomaly detection and prediction models that leverage node's proximity information and advance graph algorithms, like[10]. Query 4,5,6 targets the extraction of insights from job data. Overall, these queries are instrumental for root cause analysis of anomalous behaviors arising from the submitted jobs. By delving into job-related data, the aim is to pinpoint irregularities, understand their origins, and ultimately contribute to the reduction of anomalies in HPC operations. This approach aligns with the overarching goal of efficient management of HPC systems through data-driven analytics and insights derived from complex ODA queries.

| No. | Query | Description |
|---|---|---|
| 1 | Generate adjacency matrix, each node connected to the closest nodes in a rack. | Finds closest nodes in the same rack and constructs an adjacency matrix. |
| 2 | Generate adjacency matrix for the entire compute room, each node connected to nearest neighbors in the 3 dimensions. | Identifies nodes in proximity in the entire compute room to form an adjacency matrix. |
| 3 | Generate adjacency matrix for nodes running the same compute job. | Finds node running the same compute job and forming its adjacency matrix. |
| 4 | Average job power. | Calculates average power consumption for a specific job. |
| 5 | How many jobs are running in a particular node, over a time-period. | Count the number of jobs running on a specific node over a time period. |
| 6 | What is the min, max, average temperature when a node is in use, over a given time-period. | Computes temperature statistics of the node in use during a specified period. |

**Table 3: Selected Queries**

## 4.4 Ontology query language: SPARQL

SPARQL is the preferred choice for ontology querying due to its seamless compatibility with the RDF structure. SPARQL aligns well with RDF's graph representation, making it ideal for expressing and retrieving information from knowledge graphs. Its triple pattern matching capability allows precise matching within RDF triples, enabling users to specify complex relationships. SPARQL's expressiveness and flexibility make it a powerful tool for crafting tailored queries. Standardized by the W3C, SPARQL ensures consistency and interoperability, establishing itself as a state-of-the-art solution for RDF querying[1, 12].

## 4.5 Evaluation criteria

The evaluation of each query primarily focuses on its simplicity and conciseness. This involves a thorough examination of the complexity, indicated by the Lines of Code(LOC) required for each query. Additionally, the assessment considers the level of domain-specific knowledge necessary for executing the query effectively. A crucial aspect of the evaluation is determining the comprehensibility of each query for individuals with limited to no direct domain knowledge. Traditional metrics such as time to execution and data fetches are not applicable in this context. The knowledge graph based on the proposed ontology resides locally, while the Examon query retrieves information directly from the real Examon installation and its remote database. Consequently, the execution time won't be utilized as a comparative metric in our evaluation. Similarly, regarding data fetches, the extensive historical data in Examon makes the volume queried substantially larger than the minimal

RDF instances (described in sec.5.1) created for experimental purposes. Hence, these metrics are not considered in our evaluation approach.

# 5 EXPERIMENTAL EVALUATION

## 5.1 Experiment setup

The knowledge graph using the proposed ODA ontology is created in the TURTLE(.ttl) format. SPARQL and Examon queries are executed in a Python environment. Examon, being operational with accessible historical data, allows retrieval of genuine historical data. Examon utilizes its specific query library, ExamonQL, while RDF and SPARQL execution in Python relies on the RDFlib library.

To initiate the process, we load the .ttl ontology file and populate the RDF graph by traversing the tables of examon's historical data and selecting small batches of a few instances from each table and expressing them in the RDF triple format, thereby constructing the knowledge graph referred to as combined_graph in these queries. The PREFIX at the start of each query serves as a unique identifier for the entire ontology, with each component's identifier as its extension. The PREFIX remains consistent in all SPARQL queries and is explicitly defined as follows: "cineca_m100" is the prefix for the ontology with its base Unique Resource Identifier (URI), "rdf" is the prefix for the RDF namespace, and "xsd" is the prefix for the XML Schema namespace, used for defining datatypes in RDF. These prefixes simplify the notation in SPARQL queries by providing shorthand representations for longer URIs.

## 5.2 Query implementation

In this section, we will analyze the implementation of each query in both SPARQL and ExamonQL, providing a detailed comparison

*5.2.1 Query 1: Generate adjacency matrix, each node connected to the closest nodes in a rack and Query 2: Generate adjacency matrix for the entire compute room, each node connected to nearest neighbors in the 3 dimensions.* These two queries are centered around obtaining topological information, specifically in the context of identifying compute nodes in close physical proximity. This focus is crucial for graph-based machine learning and artificial intelligence, where precise spatial information is essential for generating adjacency matrices. It's noteworthy that these two queries are not feasible to execute using Examon due to the absence of spatial information in Examon. We present the SPARQL query aligned with the proposed ontology (Fig. 2) for further exploration. This process involves retrieving the positions of all nodes within a rack and presenting the results.

```
query = f"""SELECT ?node ?nodeId ?pos
WHERE {{
 cineca_m100:rack{rack_number} cineca_m100:
    hasNode ?node .
 ?node cineca_m100:nodeId ?nodeId .
 ?node cineca_m100:hasPosition ?pos .
}}"""
```

**Query 1 and 2: SPARQL**

The final manipulation process may differ based on different edge connectivity strategies. We combine the first two queries into a single subsection due to their similarity and shared requirements. Notably, the semantic nature of this query establishes a hierarchy, starting from identifying the target rack to its nodes and positions. SPARQL's semantic clarity enables intuitive understanding, even for individuals with limited domain knowledge familiar with the ontology and its basic concepts.

*5.2.2 Query 3: Generate adjacency matrix for nodes running the same compute job.* This query focuses on job-specific analysis and the direct linkage in the proposed ontology between job and nodes makes its implementation simpler (lower LOC count, fewer parameters and namespaces based on proposed ontology which are not specific to an ODA framework or HPC facility) than in ExamonQL. This structure can be utilized as follows by identifying the job by its "job_id" and examining its "usesNode" property to retrieve the list of nodes where this job was executed. Whereas in Examon, accessing specific data is more intricate due to the absence of direct relations between its ODA components. Retrieving particular information necessitates a deep understanding of Examon and its heterogeneous data types. Users must possess domain knowledge (covering both ODA's data types and HPC internal structure ) to identify the relevant data source, determine which data table holds the needed information, and navigate the complete ODA framework to access the necessary data.

```
query = f"""SELECT ?node ?nodeId
WHERE {{
    cineca_m100:Job{job_id} cineca_m100:usesNode
    ?node .
    ?node cineca_m100:nodeId ?nodeId .
}}"""
```

**Query 3: SPARQL**

```
#Setup for Marconi100
sq.jc.JOB_TABLES.add('job_info_marconi100')
data = sq.SELECT('*').FROM('job_info_marconi100')
    .WHERE(job_id={jobId}).TSTART({job_start_time
    }).TSTOP({job_end_time}).execute()
#create dataframe of the query result
df = pd.read_json(data)
print(df['cpus_alloc_layout'][0])
```

**Query 3: Examon**

*5.2.3 Query 4: Average job power.* Implementation of this query in SPARQL begins by identifying the nodes used and retrieving start and end times for a job. It then follows a relationship pathway from these nodes to their associated plugins and subsequently to their sensors. In this particular instance, the query selects the "total_power" sensor. Following this, the query proceeds to collect all readings from the selected sensor and apply a filter based on the job's timestamp to narrow down the readings to those within the job's specified period. Finally, the query concludes by grouping each node's values using the groupby command.

```
query = f"""SELECT ?nodeId ?unit (AVG(?powerValue
    ) AS ?averagePower)
WHERE {{
 cineca_m100:Job{job_id} cineca_m100:usesNode ?
    node ;
    cineca_m100:startTime ?startTime ;
    cineca_m100:endTime ?endTime .
```

```
 ?node cineca_m100:hasPlugin/cineca_m100:
    hasSensor ?sensor ;
    cineca_m100:nodeId ?nodeId .
 ?sensor cineca_m100:sensorName "total_power" ;
      cineca_m100:hasReading ?reading .
 ?reading cineca_m100:value ?powerValue ;
       cineca_m100:timestamp ?timestamp ;
       cineca_m100:unit ?unit .
 FILTER(?timestamp >= ?startTime && ?timestamp <=
     ?endTime)
}} GROUP BY ?nodeId"""
```

**Query 4: SPARQL**

In implementing this query in ExamonQL, we observe that the number of lines for both query types is almost the same, yet it appears more complex than the SPARQL query. The complexity arises because there is no inherent relationship between data sources in Examon, which requires the user to connect the dots, necessitating the users to be well-acquainted with each separate data source, its tables, and the contents of each table to successfully execute this query. The user has to navigate through different data sources and establish the necessary connections manually. To facilitate this process, the use of helper functions in Python becomes essential, further contributing to the complexity of the query implementation. In Examon, two sub-queries are required: one to gather job-related data and another to retrieve sensor readings. Users must integrate job information from the first sub-query into the second to obtain the final value. This multi-step process adds complexity compared to the straightforward SPARQL query.

```
def get_data(node_to_get,start_time,end_time):
  data = sq.SELECT('*') \
    .FROM('total_power').WHERE(node=node_to_get).
    TSTART(str(start_time)).TSTOP(str(end_time)).
    execute()
  value = data.df_table['value']
  return value
sq.jc.JOB_TABLES.add('job_info_marconi100')
data = sq.SELECT('*') \
    .FROM('job_info_marconi100').WHERE(job_id=
    jobId).TSTART({start_time}).TSTOP({end_time})
    .execute()
# create df of the query result
df = pd.read_json(data)
# get the allocated nodes list
dict_of_nodes = df['cpus_alloc_layout'][0]
nodes = list(dict_of_nodes.keys())
start_time = format_TS(str(df['start_time'][0]))
end_time = format_TS(str(df['end_time'][0]))
for node in nodes:
  df = get_data(node,start_time,end_time)
  avg = df.sum()/len(df)
```

**Query 4: Examon**

*5.2.4 Query 5: How many jobs are running in a particular node, over a time-period.* The implementations clearly show that the SPARQL query requires fewer lines of code (LOC) compared to the ExamonQL query. This pattern aligns with the observation in Query 4, where a single SPARQL query is used instead of two ExamonQL subqueries due to the lack of connection between separate data sources in Examon. Moreover, the semantic nature of SPARQL provides a logical structure that is easier to understand for individuals with a basic understanding of the proposed ODA ontology. In contrast, the ExamonQL implementation underscores the necessity of domain knowledge to achieve the desired output.

```
query = f"""SELECT (COUNT(?job) as ?jobCount)
WHERE {{
 ?job a cineca_m100:Job ;
 cineca_m100:usesNode cineca_m100:Node{node_id} ;
 cineca_m100:startTime ?startTime ;
 cineca_m100:endTime ?endTime .
 FILTER(?startTime <= "{end_time}"^^xsd:dateTime
    && ?endTime >= "{start_time}"^^xsd:dateTime)
}}"""
```

**Query 5: SPARQL**

```
def get_nodes_list(jobId,time_period):
  data = sq.SELECT('*')\
  .FROM('job_info_marconi100')\
  .WHERE(job_id=str(jobId))\
  .TSTART(time_period[0]).TSTOP(time_period[1]).
    execute()
  # create df of the query result
  df = pd.read_json(data)
  # get the allocated nodes list
  dict_of_nodes = df['cpus_alloc_layout'][0]
  try: nodes = list(dict_of_nodes.keys())
  except: pass
  # create df of the query result
  df = pd.read_json(data)
  df['cpus_alloc_layout'][0]
  nodes = list(dict_of_nodes.keys())
  return nodes
# Setup for Marconi100
sq.jc.JOB_TABLES.add('job_info_marconi100')
data = sq.SELECT('*') \
    .FROM('job_info_marconi100').TSTART({
    start_time}).TSTOP({end_time}).execute()
df = pd.read_json(data)
job_ids = df['job_id'].to_numpy()
count = 0
for job_id in job_ids:
  try: nodes_list = get_nodes_list(job_id,
    time_period)
       count += nodes_list.count(node_to_check)
  except: pass
```

**Query 5: Examon**

*5.2.5 Query 6: What is the min, max, average temperature when a node is in use, over a given time-period.* In the implementation of this query, we can see that ExamonQL requires a lot more lines of code(LOC) as compared to SPARQL. Additionally, in the case of ExamonQL for this query, the necessity for four sub-queries to obtain the final results further emphasizes the increased complexity in comparison to the more concise SPARQL implementation.

```
query = f"""SELECT ?nodeId (AVG(?temperature) as
    ?avgTemperature) (MIN(?temperature) as ?
    minTemperature) (MAX(?temperature) as ?
    maxTemperature)
WHERE {{
```

```
 ?job rdf:type cineca_m100:Job ;
    cineca_m100:startTime ?jobStart ;
    cineca_m100:endTime ?jobEnd ;
    cineca_m100:usesNode ?node .
 ?node cineca_m100:hasPlugin/cineca_m100:
    hasSensor ?sensor ;
    cineca_m100:nodeId ?nodeId .
 ?sensor cineca_m100:sensorName "temperature" ;
    cineca_m100:hasReading ?reading .
 ?reading cineca_m100:value ?temperature ;
    cineca_m100:timestamp ?timestamp ;
    cineca_m100:unit ?unit .
 FILTER(?jobStart <= "{end_time}"^^xsd:dateTime
    && ?jobEnd >= "{start_time}"^^xsd:dateTime)
}}GROUP BY ?nodeId"""
```

**Query 6: SPARQL**

```
def get_nodes_list(jobId,time_period):
  data = sq.SELECT('*') \
    .FROM('job_info_marconi100').WHERE(job_id=str
    (jobId))\
    .TSTART(time_period[0]).TSTOP(time_period[1])
    .execute()
  # create df of the query result
  df = pd.read_json(data)
  # get the allocated nodes list
  dict_of_nodes = df['cpus_alloc_layout'][0]
  try: nodes = list(dict_of_nodes.keys())
  except: pass
  # create df of the query result
  df = pd.read_json(data)
  df['cpus_alloc_layout'][0]
  nodes = list(dict_of_nodes.keys())
  return nodes
# Setup for Marconi100
sq.jc.JOB_TABLES.add('job_info_marconi100')
data = sq.SELECT('*').FROM('job_info_marconi100')
    .TSTART({start_time}).TSTOP({end_time})\
    .execute()
df = pd.read_json(data)
job_ids = df['job_id'].to_numpy()
node_used_in_job_list = []
for job_id in job_ids:
  try: nodes_list = get_nodes_list(job_id,
    time_period)
    if (node_to_check in nodes_list):
      print(job_id,nodes_list)
      node_used_in_job_list.append(job_id)
  except: pass
def get_data(node_to_get,metric,start_time,
    end_time):
  data = sq.SELECT('*').FROM(metric).WHERE(node=
    node_to_get).TSTART(str(start_time)).TSTOP(
    str(end_time)).execute()
  value = data.df_table['value']
  return value
def get_job_time(jobId):
  data=sq.SELECT('*').\
  FROM('job_info_marconi100')\
  .WHERE(job_id=str(jobId),node=node_to_check)\
  .TSTART({start_time}).TSTOP({end_time})\
  .execute()
  df = pd.read_json(data)
  start_time= format_TS(str(df['start_time'][0]))
  end_time= format_TS(str(df['end_time'][0]))
  return start_time,end_time
each_job_df = []
for job in node_used_in_job_list:
  start_time,end_time = get_job_time(job)
  try:
    df = get_data(node_to_check,metric,start_time
    ,end_time)
    each_job_df.append((max(df),min(df),(df.sum()
    /len(df))))
  except: print("error")
```

**Query 6: Examon**

## 6 DISCUSSION

The evaluation of SPARQL queries against ExamonQL provides valuable insights into their efficiency and usability for querying topological information and conducting job-specific analyses within HPC environments (see sec. 5.2). In queries 1 and 2, SPARQL's semantic clarity and alignment with the proposed ontology enable intuitive querying, starting from rack identification to node positions. In contrast, Examon lacks spatial information, rendering such queries unfeasible in ExamonQL. For queries 3, 4, 5, and 6, the direct linkage between jobs and their utilized nodes in the proposed ontology simplifies query implementation, resulting in fewer lines of code and reduced complexity compared to ExamonQL. Additionally, SPARQL's filtering capabilities lead to a more concise and logical query structure, whereas ExamonQL's fragmented queries lead to increased complexity.

Overall, SPARQL consistently demonstrates advantages in efficiency and usability across all six queries, offering a structured framework that simplifies query development and comprehension. In contrast, ExamonQL's manual connection requirements and fragmented querying pose challenges for users, necessitating a deeper understanding of the underlying connectivity between different data sources.

## 7 CONCLUSION

In this manuscript, we presented an ontology for ODA and a comparative analysis with state-of-the-art ODA methods. The comparative analysis of complex ODA queries implemented in Examon and SPARQL sheds light on the practical applicability of SPARQL, showcasing its efficiency and clarity in query execution(fewer LOC and less domain knowledge requirements). SPARQL's semantic nature allows users to comprehend queries by following the logical structure outlined in the proposed ontology. With even basic knowledge of the proposed ontology, its classes, and relationships, users can easily grasp the query's intent. This feature enhances accessibility and comprehension without necessitating extensive domain expertise. SPARQL query seamlessly aligns with the inherent relations in the HPC data, making queries transparent and aiding a straightforward understanding. Future work involves further refining the ontology, assessing capabilities with more complex queries, and converting historical Examon datasets into RDF format for deployment in graph databases for further comparative analysis.

## ACKNOWLEDGMENT

This research was partly supported by the EuroHPC EU Regale project (g.a. 956560), the HE EU Graph-Massivizer project (g.a. 101093202), and also the Spoke "FutureHPC & BigData" of the ICSC – Centro Nazionale di Ricerca in "High Performance Computing, Big Data and Quantum Computing", funded by European Union – NextGenerationEU. We also express our gratitude to CINECA for their collaboration and providing access to their machines.

## REFERENCES

[1] Waqas Ali, Muhammad Saleem, Bin Yao, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. 2022. A survey of RDF stores & SPARQL engines for querying knowledge graphs. *The VLDB Journal* 31a, 3 (May 1 2022), 1–26. https://doi.org/10.1007/s00778-021-00711-3

[2] Andrea Bartolini, Francesco Beneventi, Andrea Borghesi, Daniele Cesarini, Antonio Libri, Luca Benini, and Carlo Cavazzoni. 2019. Paving the Way Toward Energy-Aware and Automated Datacentre. In *Proceedings of the 48th International Conference on Parallel Processing: Workshops* (Kyoto, Japan) *(ICPP 2019)*. Association for Computing Machinery, New York, NY, USA, Article 8, 8 pages. https://doi.org/10.1145/3339186.3339215

[3] Andrea Borghesi, Carmine Di Santi, Martin Molan, Mohsen Seyedkazemi Ardebili, Alessio Mauri, Massimiliano Guarrasi, Daniela Galetti, Mirko Cestari, Francesco Barchi, Luca Benini, Francesco Beneventi, and Andrea Bartolini. 2023. M100 ExaData: a data collection campaign on the CINECA's Marconi100 Tier-0 supercomputer. *Scientific Data* 10, 1 (18 May 2023), 288. https://doi.org/10.1038/s41597-023-02174-3

[4] Gabriel G. Castañé, Huanhuan Xiong, Dapeng Dong, and John P. Morrison. 2018. An ontology for heterogeneous resources management interoperability and HPC in the cloud. *Future Generation Computer Systems* 88 (2018), 373–384. https://doi.org/10.1016/j.future.2018.05.086

[5] Oscar Corcho, David Chaves-Fraga, Jhon Toledo, Julián Arenas-Guerrero, Carlos Badenes-Olmedo, Mingxue Wang, Hu Peng, Nicholas Burrett, José Mora, and Puchao Zhang. 2021. A High-Level Ontology Network for ICT Infrastructures. In *The Semantic Web – ISWC 2021*, Andreas Hotho, Eva Blomqvist, Stefan Dietze, Achille Fokoue, Ying Ding, Payam Barnaghi, Armin Haller, Mauro Dragoni, and Harith Alani (Eds.). Springer International Publishing, Cham, 446–462.

[6] Pouya Kousha, Vivekananda Sathu, Matthew Lieber, Hari Subramoni, and Dhabaleswar K. Panda. 2023. Democratizing HPC Access and Use with Knowledge Graphs. In *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis* (<conf-loc>, <city>Denver</city>, <state>CO</state>, <country>USA</country>, </conf-loc>) *(SC-W '23)*. Association for Computing Machinery, New York, NY, USA, 243–251. https://doi.org/10.1145/3624062.3624094

[7] Chunhua Liao, Pei-Hung Lin, Gaurav Verma, Tristan Vanderbruggen, Murali Emani, Zifan Nan, and Xipeng Shen. 2021. HPC Ontology: Towards a Unified Ontology for Managing Training Datasets and AI Models for High-Performance Computing. In *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*. 69–80. https://doi.org/10.1109/MLHPC54614.2021.00012

[8] Brian McBride. 2004. *The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS*. Springer Berlin Heidelberg, Berlin, Heidelberg, 51–65. https://doi.org/10.1007/978-3-540-24750-0_3

[9] Dejan Milojicic, Paolo Faraboschi, Nicolas Dube, and Duncan Roweth. 2021. Future of HPC: Diversifying Heterogeneity. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*. 276–281. https://doi.org/10.23919/DATE51398.2021.9474063

[10] Martin Molan, Junaid Ahmed Khan, Andrea Borghesi, and Andrea Bartolini. 2023. Graph Neural Networks for Anomaly Anticipation in HPC Systems. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 239–244.

[11] M. Ott, W. Shin, and et al. 2020. Global Experiences with HPC Operational Data Measurement, Collection and Analysis. In *2020 IEEE International Conference on Cluster Computing*.

[12] Bastian Quilitz and Ulf Leser. 2008. Querying Distributed RDF Data Sources with SPARQL. In *The Semantic Web: Research and Applications*, Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 524–538.

[13] Woong Shin, Vladyslav Oles, Ahmad Maroof Karimi, J. Austin Ellis, and Feiyi Wang. 2021. Revealing Power, Energy and Thermal Dynamics of a 200PF Pre-Exascale Supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 12, 14 pages. https://doi.org/10.1145/3458817.3476188

[14] Lauri Tuovinen and Jaakko Suutala. 2021. Ontology-based Framework for Integration of Time Series Data: Application in Predictive Analytics on Data Center Monitoring Metrics. 151–161. https://doi.org/10.5220/0010650300003064

[15] Wikipedia. 2021. CINECA — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=CINECA&oldid=954269846. [Online; accessed 04-December-2021].

# An Extensive Characterization of Graph Sampling Algorithms

S. Haleh S. Dizaji
University of Klagenfurt
Klagenfurt am Wörthersee, Austria
Seyedehhaleh.Seyeddizaji@aau.at

Jože M. Rožanec
Jožef Stefan Institute
Ljubljana, Slovenia
joze.rozanec@ijs.si

Reza Farahani
University of Klagenfurt
Klagenfurt am Wörthersee, Austria
reza.farahani@aau.at

Dumitru Roman
SINTEF
Oslo, Norway
dumitru.roman@sintef.no

Radu Prodan
University of Klagenfurt
Klagenfurt am Wörthersee, Austria
radu.prodan@aau.at

## ABSTRACT

While graph sampling is key to scalable processing, little research has tried to thoroughly compare and understand how it preserves features such as degree, clustering, and distances dependent on the graph size and structural properties. This research evaluates twelve widely adopted sampling algorithms across synthetic and real datasets to assess their qualities in three metrics: degree, clustering coefficient (CC), and hop plots. We find the random jump algorithm to be an appropriate choice regarding degree and hop-plot metrics and the random node for CC metric. In addition, we interpret the algorithms' sample quality by conducting correlation analysis with diverse graph properties. We discover eigenvector centrality and path-related features as essential features for these algorithms' degree quality estimation, node numbers (or the size of the largest connected component) as informative features for CC quality estimation and degree entropy, edge betweenness and path-related features as meaningful features for hop-plot metric. Furthermore, with increasing graph size, most sampling algorithms produce better-quality samples under degree and hop-plot metrics.

## CCS CONCEPTS

• **Applied computing** → *Computer forensics*; *System forensics*;

## KEYWORDS

Graph sampling algorithms, Scalable graph processing

**ACM Reference Format:**
S. Haleh S. Dizaji, Jože M. Rožanec, Reza Farahani, Dumitru Roman, and Radu Prodan. 2024. An Extensive Characterization of Graph Sampling Algorithms. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3629527.3652899

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652899

## 1 INTRODUCTION

Graphs offer a flexible approach to modeling connected components and carry useful information about relationships of the structured data. However, accessing or processing full graphs in large-scale scenarios is infeasible or poses considerable challenges. For example, computing measures such as shortest paths, clusterings, or betweenness centrality (BC) become impractical [12] on large graphs. In such scenarios, *graph sampling* [12] is a popular remedy that allows for estimating these properties from a small fraction of its nodes and edges [25]. In addition, sampling can benefit machine learning tasks, with training more effectively on smaller fractions of the data. In particular, it can directly influence the robustness [3] and performance [1] of graph neural networks.

As the graph sampling algorithms become more extensive, studying their behavior becomes more demanding, as they perform differently depending on desired quality metrics and graphs. Unfortunately, literature remains scarce, and few works address this area, considering the limited amount of synthetic or real graphs. Furthermore, they do not provide an in-depth analysis of sampling quality considering graph size and structural features.

To bridge this void, we compare twelve graph sampling methods across around 2900 synthetic graphs of six types and twelve real datasets. We assess them using three metrics considered in the literature [12, 27], i.e., degree, clustering coefficient (CC), and hop-plots, to evaluate the qualities of samples regarding the original graphs. We quantify the dependency of these properties on graph features (77 features) and find the most relevant ones for each algorithm and metric. We uncover some important dependencies and highlight the most relevant features for different algorithms regarding each metric. In addition, we evaluate algorithms on small and large real graphs, confirming some of the relevant features obtained for synthetic ones.

The paper has seven sections. Section 2 reviews relevant sampling algorithms. Section 3 introduces related studies to our research. Section 4 defines the metrics used for evaluating samplings' result quality. Section 5 explains the experimental setup, including datasets, and experimental settings. Section 6 analyzes the results. Finally, section 7 concludes the paper and outlines future research.

## 2 GRAPH SAMPLING ALGORITHMS

We characterize graph sampling algorithms for static networks under three categories: node, edge, and traversal-based sampling [12]. This paper contributes to the state-of-the-art by investigating the

sampling qualities of twelve popular algorithms of the three categories under various graph properties.

## 2.1 Node-based sampling

Node-based methods are most intuitive but only weakly preserve properties of specific graph types [2, 22], possibly losing connectivity [9]. *Random node (RN)* can preserve the CC for some graphs [9] and the degree distribution for random graphs [22], however poorly preserves the power-law degree distribution [14, 22] and average path length (APL) for non-small samples. *Random degree node (RDN)* applies probabilistic node selection proportional to the degrees [12], but loses degree distribution by creating bias over high-degree nodes [22]. *Random PageRank node* mitigates this bias [14] using nodes PageRank scores [20]. *Node sampling with contraction* reduces the graph's size by randomly removing nodes [6].

## 2.2 Edge-based sampling

Edge-based sampling can preserve edge-dependent properties, such as path length [2]. On the other hand, primary edge-based samplers have bias over high-degree nodes and poorly preserve some properties, such as connectivity and clustering. *Random edge (RE)* has poor preservation of graph structure (higher APLs for larger samples and lower CC). *Random node edge (RNE)* randomly selects a node and its edge [12]. RN selection mitigates bias over high-degree nodes [12]; however it can generate sparse graphs [14] due to limited edge selection. To solve this problem, *hybrid sampling* performs RNE or RE steps probabilistically [12], resulting in less bias towards high-degree nodes than RE. *Induced random edge (IRE)*, an extension of RE, performs an induction step by adding all edges between selected nodes in RE, collecting more information and better preserving the topological properties [2]. *Edge sampling with contraction* generates samples by randomly removing an edge and merging nodes previously joined by that edge [6].

## 2.3 Traversal-based sampling

Traversal-based methods improve the performance of RN and RE methods by capturing topological information of graph [2, 6].

*Random traversal methods. Random walk (RW)* performs sampling initialized from one seed node [21] with a better degree distribution estimation [18], but can get stuck in a graph region. To overcome this problem, *random jump (RJ)* jumps to a random node with some probability. *Metropolis-Hastings random walk (MHRW)* selects the neighboring nodes in RW proportional to degree ratios [23], but fails to estimate the degree distribution well [18]. *Multiple independent random walkers* avoid sampling from a specific region [6], [4], resulting in higher estimation errors [17].

*Neighborhood exploration methods. Snowball (SB)* traverses the graph by selecting a fixed number of neighbors of the current node set [5, 6], which preserves CC for certain graphs [9], but suffers from boundary bias [9], underestimating power-low degree distribution exponent and lower APL [9]. *Forest fire (FF)* adapted from the evolution network model [10] mitigates the local sampling problem of SB with the neighborhood size following a geometric distribution [6] with a bias over high-degree nodes and getting stuck in isolated clusters regions [14]. *Frontier sampling (FS)* performs probabilistic node selection from the current set according to its degree and replaces it randomly with one of its neighbors [17]; however, increasing the number of seed nodes (infinitely) results in uniform node and edge distribution [6]. *Expansion sampling (XS)* aims to preserve some graph community structure [13, 27] by starting from a random seed and traversing the neighborhood by selecting the node maximizing out-links of the current sample. *Rank degree (RD)* preserves community structure [27] by ranking the node neighborhood by degrees [24], randomly selecting a node from a seed set and its top-k neighbors as sample edges and replacing the seed set with them. *Tight sampling (TS)* mitigates the local sampling of SB trying to preserve local clusters around seed nodes [8]. *List sampling (LS)* tries to solve poor neighborhood exploration using a list of currently sampled nodes' neighbors [28] and has a better APL estimation on graphs with high CC [27].

# 3 RELATED WORK

We summarize the studies for graph sampling algorithms analysis in two sections: analytical and numerical evaluations.

## 3.1 Analytical evaluations

Stumpf and Wiu [22] analyzed RN on random, exponential and scale-free graphs and Lee et al. [9] studied RN and RE on Albert-Barabasi (AB) and real graphs. They characterized the degree distribution of samples dependent on the original graph degree distribution and sampling rate. Illenberger and Flötteröd [7] analyzed SB algorithm on Erdos-Renyi (ER) and real graphs and concluded that the original graphs' mean degree, degree correlation, and CC estimation quality decrease with the increasing variance of the original graph degree distribution. Ribeiro and Towsley [18] analyzed RN, RE, RW, and MHRW, estimating the graph degree distribution based on the unbiased Horvitz-Thompson estimator dependent on sample degrees and distributions and verified on large real graphs.

*Limitations.* While providing accurate estimations, these analyses study limited sampling algorithms and synthetic graphs and do not consider various graph properties. We analyze several algorithms (including updated algorithms) under six synthetic and twelve real graphs, considering several graph features.

## 3.2 Numerical evaluation

Leskovec and Faloutsos [12] evaluated ten node, edge, and traversal-based algorithms under scale-down and back-in-time samplings using nine metrics (i.e., degree, CC, connected components sizes, hop-plots, and singular values distributions) over four real graphs concluding that traversal-based algorithms yield better results for static graphs. Yoon et al. [26] evaluated RW under quality metrics, i.e., degree distribution, CC, and degree-degree correlation for Albert-Barabasi (AB) and three real graphs and found for high power-law degree distribution exponents, RW preserves most topological properties and reported deviations in small samples' degree distribution exponents with increasing the exponent. Lee et al. [9] studied RN, RE, and SB under degree, BC, APL, assortativity, and CC and found very different quantities of these properties for these

algorithms. Zhang et al. [29] studied fourteen samplers of all categories using random and real graphs under numerical quality metrics (degree, BC, and hop-plots distributions), visualization, and execution time and discovered that the algorithm's performance depends on graph type, size, and measured property. Yousuf et al. [27] evaluated five traversal-based algorithms for twelve large real and three synthetic graphs, i.e., forest fire model (FFM), Watts-Strogatz (WS) and mixed model under degree, CC, and path length distributions, global CC (GCC), assortativity and modularity and analyzed their performance for various graph types and properties, and concluded that algorithms aggressively exploring the sample node's neighborhood better preserve structural properties and the selection of high-degree nodes is beneficial.

*Limitations.* Despite several studies, none characterize these sampling algorithms thoroughly under diverse graph properties. We try to fill this gap by analyzing correlations between quality metrics and graphs' size and topological features on six synthetic data types and twelve real graphs.

## 4 SAMPLING EVALUATION METRICS

We analyze the performance of a sampling algorithm under quality metrics, assessing the similarity of the sample to the original graph under a desired property to preserve.

### 4.1 Graph Properties

We considered three popular structural graph properties as sampling quality metrics.

(1) *Degree distribution* captures the overall degree structure in the graph in terms of the number of edges connected to each node.
(2) *CC distribution* evaluates the clustering property around every node formulated as the number of closed triangles divided by the possible (closed or open) number of triangles.
(3) *Hop-plot distribution* evaluates the closeness of interconnected nodes (similar to the shortest path) [12, 15] by counting the number of pairs separated by a maximum number of hops.

### 4.2 Distributions Divergence

Among the different distribution divergence metrics in the literature, we consider the *Kolmogorov-Smirnov D-statistic* metric used in previous studies [12, 29] for analyzing samplings:

$$KS = |\max\left(F_G(x) - F_{Gs}(x)\right)|,$$

where $G$ and $Gs$ are original and sample graphs and $F_G(.)$ is the cumulative distribution function of graph $G$. We normalize the distributions to be independent of graph size and capture structural properties, similar to [12]. We analyze sampling algorithms using three quality metrics based on this definition: degree (D3), CC (C2D2), and hop-plots (HPD2) distribution divergences.

## 5 EXPERIMENTAL DESIGN

We describe the extracted graph features, datasets, and experimental settings in our experiments.

| *Type* | $\|E\|$ | $\|G\|$ | $\overline{Deg}$ | $D$ | $\overline{CC}$ | $\overline{EBC}$ | $\frac{\|N\|}{\|E\|}$ | $H(Deg)$ | $H(CC)$ | $\overline{EIC}$ | $Dia$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| AB | 196 ∼ 640,000 | 460 | 115 | 0.13 | 0.19 | 2868 | 0.2 | 3.60 | 1.58 | 0.04 | 4.70 |
| ER | 1 ∼ 800,479 | 560 | 111 | 0.12 | 0.12 | 2298 | 1.67 | 2.71 | 0.65 | 0.04 | 5.86 |
| WS | 200 ∼ 800,000 | 460 | 135 | 0.15 | 0.23 | 6322 | 0.20 | 2.60 | 1.51 | 0.04 | 13.02 |
| PLC | 196 ∼ 5991 | 480 | 5 | 0.02 | 0.32 | 3552 | 0.42 | 1.84 | 2.32 | 0.03 | 6.75 |
| FFM | 104 ∼ 1,801,233 | 464 | 164 | 0.22 | 0.45 | 6013 | 0.51 | 2.77 | 1.99 | 0.03 | 14.28 |
| SBM | 316 ∼ 404,879 | 475 | 117 | 0.13 | 0.29 | 1726 | 0.06 | 3.96 | 2.43 | 0.04 | 3.46 |

**Table 1: Characteristics of synthetic graphs. (|G|: number of graphs).**

| *Dataset* | $\|N\|$ | $\|E\|$ | $\overline{CC}$ | $H(deg)$ | $H(CC)$ | $CC_{var}$ | $EIC_{max}$ | $Dia$ |
|---|---|---|---|---|---|---|---|---|
| Bio | 924 | 3239 | 0.88 | 2.62 | 2.62 | 0.122 | 0.32 | 10 |
| Email | 1005 | 16,064 | 0.54 | 4.32 | 3.4 | 0.063 | 0.17 | 7 |
| Pow-1138 bus | 1138 | 1458 | 0.09 | 1.68 | 1.09 | 0.056 | 0.41 | 31 |
| Euroroad | 1174 | 1417 | 0.02 | 1.39 | 0.42 | 0.007 | 0.22 | 62 |
| Soc-Wiki vote | 889 | 2914 | 0.15 | 2.7 | 2.41 | 0.050 | 0.29 | 13 |
| Tech-ISP | 2113 | 6632 | 0.25 | 2.55 | 2.32 | 0.113 | 0.20 | 12 |
| Tech-Topology | 34,761 | 107,720 | 0.29 | 1.87 | 1.64 | 0.167 | 0.33 | 10 |
| Tech-Gnutella | 62,586 | 147,892 | 0.005 | 2.08 | 0.18 | 0.003 | 0.04 | 11 |
| Tech-Caida | 190,914 | 607,610 | 0.16 | 2.58 | 2.06 | 0.072 | 0.07 | 26 |
| Cit-Cora | 23,166 | 89,157 | 0.27 | 2.92 | 2.91 | 0.082 | 0.14 | 20 |
| Cit-HepTh | 27,769 | 352,285 | 0.31 | 4.14 | 3.40 | 0.049 | 0.26 | 15 |
| Cit-HepPh | 34,546 | 420,877 | 0.28 | 4.14 | 3.40 | 0.043 | 0.11 | 14 |

**Table 2: Characteristics of real-world graphs.**

### 5.1 Graph features

We considered several graph size and topology features, their statistics (minimum, maximum, median, mean, variance), and the features' calculation time. These features consist of node and edge numbers ($|N|$ and $|E|$), degree (*deg*), *CC*, and *GCC*, degree and CC entropy ($H(.)$), degree assortativity, density ($D$), node and edge BC (*NBC* and *EBC*), number and sizes of connected components (*ConCS*), eccentricity (*ECC*), eigenvector (*EIC*), PageRank and farness (*FC*) centralities, maximum spanning tree degrees (*DMST*), diameter (*dia*) and shortest path length (*SPL*).

### 5.2 Datasets

*Synthetic graphs.* We generated around 2900 graphs of six types, i.e., AB, WS, ER, power-low-cluster (PLC), stochastic block model (SBM), and FFM with $|N|$ of 100 ∼ 2000, summarized in Table 1. These graph types have different properties, i.e., scale-free (AB and PLC), clustering (WS and PLC), community structure (SBM), evolving pattern (FF), and theoretical implications (ER). For further analysis, we extracted 77 graph features (size and topology) and reported average values of the most relevant ones in Table 1.

*Real graphs.* We considered twelve publicly available[1] [11, 19] real graphs of various sizes of around 1000 to 190.000 nodes and categories, including power, biological, email, infrastructure, social, citation, and technology (Internet service provider (ISP)) graphs. Table 2 represents their characteristics and relevant features.

### 5.3 Experimental setup

We conducted two sets of sampling experiments in our analysis:

(1) *Small synthetic graphs* finding correlations between sampling algorithms' performance and graphs' features;
(2) *Small and large real-world graphs* investigating algorithms' behavior according to the correlation results.

[1] http://konect.cc/networks/

We considered sampling rates of 0.1 and 0.3, representing the approximate percentage of graph nodes sampled from the graph. We conducted each sampling experiment for five iterations and reported the average results over different sampling rates, graph types, and sampling iterations. We used the *Pearson correlation coefficient* $\rho$ [16] for quantifying the relationship between the graph quality metrics introduced in Section 4.1 and graph features.

# 6 EVALUATION RESULTS

We provide analysis and evaluations on synthetic and real graphs.

## 6.1 Synthetic graphs

We summarize the results for the three quality metrics for four graph types and analyze their dependency on graph properties.

*6.1.1 Degree distribution divergence.* Figure 1(a) compares only four graph types (AB, ER, WS, and SBM) with similar average densities (see Table 1), illustrating the relatively better performance of most algorithms on AB graphs. XS and FF are the best algorithms.

*Correlation analysis.* Figure 2(a) represents the highly correlated sampling algorithms and graph features (i.e., $|\rho| > 0.5$), including only some statistics of features. The highest correlated features regardless of algorithms are $EIC_{\max}$, $H(deg)$, $CC_{var}$ and $EBC_{med}$. We also observed a higher correlation of path-related features (*FC*, *SPL*, *dia* and *ECC*) with traversal-based algorithms, representing better traversing and degree distribution preservation in graphs with higher path lengths. This feature also impacts RE. *EBC*, $CC_{var}$ also are more relevant to traversal algorithms, with *EIC* and $H(deg)$ being relevant to most traversal algorithms (indicating their poor degree preservation on graphs with highly randomized degrees, such as SBM graphs (Figure 1(a))). Density is more relevant to FF and RJ. There is also a high relevance of *DMST* to RNE.

*6.1.2 Clustering coefficient distribution divergence.* Figure 1(b) represents a better sampling quality in C2D2 than in D3 metric, with better results for WS graphs. These results indicate the better CC preservation of RN and RD for most cases.

*Correlation analysis.* Figure 2(b) represents the highly correlated graph features with sampling algorithms' C2D2 results. $H(deg)$, $|N|$, $ConCS_{max}$ and *NBC* are the most relevant feature for most algorithms. $|N|$, $ConCS_{max}$, and *PRC* are more correlated with node-based algorithms i.e., RN and RNE. $H(deg)$ and *DMST* are most relevant to RD, MHRW, and FS traversal algorithms that are biased over higher degree nodes. *NBC* is important for edge-based algorithms (RE, IRE, and RNE) and RDN.

*6.1.3 Hop-plot distribution divergence.* Figure 1(c) reveals FF as the best algorithm for almost all four graph types. RJ and MHRW have a low HPD2 for some graph types.

*Correlation analysis.* Figure 2(c) reveals some interesting high HPD2 correlations with path-related features and *EBC*. Decreasing path-related features results in lower HPD2 for most algorithms, rising from lost connectivity by sampling (except for SB and FS algorithms). We observed the same pattern for *EBC* and *NBC*. Whereas *D*, *CC*, $H(deg)$, *DMST*, *deg* and $|E|/|N|$ are negatively correlated with most algorithms, however, they reverse impact FS and SB. This indicates better distance preservation by decreasing distances in dense or highly clustered graphs.

## 6.2 Real-world graphs

*6.2.1 Degree distribution divergence.*

*Small graphs.* Tables 9 and 3 represent that RJ and FF are the best algorithms. Most traversal-based algorithms have D3 under 0.2 (with below 0.1 for FF) for Road and Bus datasets having high path lengths, high $EIC_{\max}$ and $EBC_{med}$, low $H(deg)$ and density relevant to most algorithms (Figure 2). RJ has a low D3 for the Bio graph with a high $CC_{var}$ and rather high $EIC_{max}$ relevant to RJ. Almost all algorithms have high D3 for the Email dataset, having a low $EIC_{\max}$ and $EBC_{med}$, and high $H(deg)$ relevant to all samplings.

| | FF | XS | RJ | FS | MHRW | RN | RNE | RE | IRE | RDN | SB | RD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Email | 0.21 | 0.44 | 0.23 | 0.74 | 0.32 | 0.67 | 0.84 | 0.52 | 0.23 | 0.26 | 0.65 | 0.25 |
| Bio | 0.18 | 0.25 | 0.12 | 0.38 | 0.25 | 0.74 | 0.83 | 0.43 | 0.15 | 0.20 | 0.61 | 0.18 |
| Bus | **0.05** | **0.09** | 0.13 | 0.12 | 0.13 | 0.65 | 0.64 | 0.46 | 0.33 | 0.52 | 0.26 | 0.55 |
| Road | **0.06** | 0.18 | 0.19 | 0.13 | 0.20 | 0.78 | 0.80 | 0.64 | 0.54 | 0.72 | 0.41 | 0.68 |
| Wiki | 0.15 | 0.31 | 0.16 | 0.37 | 0.22 | 0.61 | 0.71 | 0.39 | 0.18 | 0.17 | 0.39 | 0.25 |
| ISP | 0.24 | 0.37 | 0.17 | 0.22 | 0.22 | 0.55 | 0.60 | 0.32 | 0.18 | 0.20 | 0.26 | 0.26 |

**Table 3: Average D3 for small real-world graphs**

*Large graphs.* Large graphs revealed similar and different patterns. Overall, RJ, IRE and RD perform better than other samplers for large graphs (tables 4 and 9), where RJ is consistent with small-scale results. FS has a very low D3 for Topology network with high $EIC_{max}$ and $CC_{var}$. FF has a D3 of 0.1 for the HepPh dataset (and 0.12 for Cora) with a lower $EIC_{min}$ (opposite for Topology) and high D3 for Gnutella, with low $EIC_{\max}$ and $CC_{var}$, consistent with our findings. Therefore, FF is a better choice for citation than technology graphs. RJ and IRE produce good-quality samples for Cora, Caida and HepTh. Cora and HepTh have a very low $EIC_{\min}$ (also Caida) relevant to these algorithms. In addition, Cora and Caida have higher diameters, correlated with them. RDN better estimates the degree property of HepTh with a rather high $EIC_{\max}$.

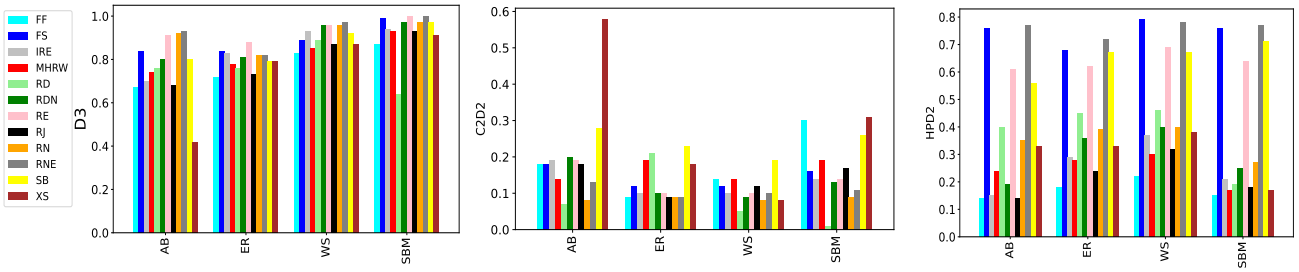
*6.2.2 Clustering coefficient distribution divergence.*

*Small graphs.* According to Table 5, RN and RNE have the best results (RN is consistent with synthetic data). Most algorithms can better capture the CC property of Bus, Wiki and ISP networks. It is interpretable for ISP network with more nodes and higher $ConCS_{max}$ relevant to C2D2 of most samplers.

*Large graphs.* Table 6 illustrates the best results for RN and RNE (as in small-scale). RN has a perfect CC preservation with a maximum C2D2 of 0.01. For Gnutella, most algorithms very well preserve the *CC* property, having low $H(CC)$ and rather high $|N|$ relevant to C2D2. Additionally, this table represents poor CC preservation of most algorithms on Caida and Topology datasets.

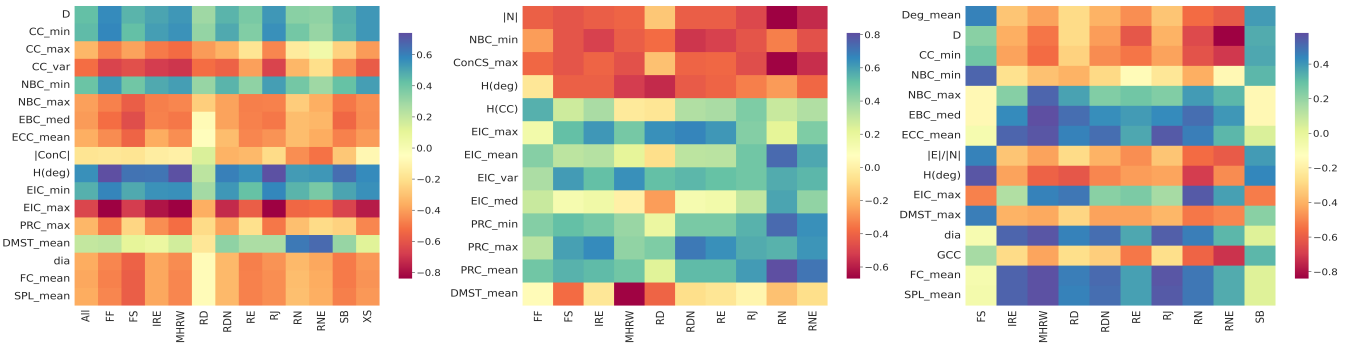
*6.2.3 Hop-plot distribution divergence.*

*Small graphs.* Table 7 represents the poor sample quality by almost all algorithms regarding HPD2 on small real graphs, except for SB in Wiki. On average (Table 9), XS, RJ and FF has relatively better results (FF performs well on synthetic (syn) graphs). We also




(a) D3. (b) C2D2. (c) HPD2.

**Figure 1: Average synthetic graph quality metric results.**




(a) D3. (b) C2D2. (c) HPD2.

**Figure 2: Correlation matrix with graph features.**

| | FF | FS | IRE | MHRW | RD | RDN | RE | RJ | RN | RNE | SB | XS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Gnutella | 0.28 | 0.24 | 0.27 | 0.29 | 0.35 | 0.24 | 0.36 | 0.24 | 0.56 | 0.44 | 0.26 | 0.33 |
| HepPh | **0.10** | 0.77 | 0.14 | 0.62 | 0.17 | 0.14 | 0.88 | 0.16 | 0.56 | 0.92 | 0.63 | 0.25 |
| HepTh | 0.17 | 0.71 | 0.12 | 0.55 | **0.08** | 0.11 | 0.83 | 0.12 | 0.51 | 0.89 | 0.57 | 0.29 |
| Cora | 0.12 | 0.43 | 0.11 | 0.39 | **0.10** | 0.17 | 0.69 | **0.10** | 0.57 | 0.79 | 0.41 | 0.14 |
| Caida | 0.24 | 0.17 | **0.10** | 0.20 | 0.14 | 0.18 | 0.55 | **0.09** | 0.55 | 0.71 | 0.24 | 0.20 |
| Topology | 0.27 | **0.04** | 0.25 | 0.34 | 0.18 | 0.28 | 0.44 | 0.19 | 0.65 | 0.59 | 0.31 | 0.29 |

**Table 4: Average D3 for large real-world graphs.**

| | FF | XS | RJ | FS | MHRW | RN | RNE | RE | IRE | RDN | SB | RD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Email | 0.3 | 0.45 | 0.26 | 0.27 | 0.23 | 0.11 | 0.16 | 0.29 | 0.28 | 0.29 | 0.23 | 0.34 |
| Bio | 0.27 | 0.36 | 0.23 | 0.26 | 0.31 | **0.10** | 0.13 | 0.21 | 0.19 | 0.24 | 0.24 | 0.34 |
| Bus | 0.29 | 0.13 | **0.10** | 0.16 | 0.45 | **0.07** | **0.08** | **0.08** | **0.07** | **0.07** | 0.26 | 0.24 |
| Road | 0.15 | 0.29 | 0.12 | 0.15 | 0.38 | **0.06** | **0.07** | 0.12 | 0.12 | 0.11 | 0.26 | 0.27 |
| Wiki | 0.33 | **0.08** | 0.15 | 0.15 | 0.28 | **0.08** | **0.09** | **0.09** | **0.09** | **0.10** | 0.12 | 0.19 |
| ISP | **0.04** | **0.04** | **0.05** | **0.04** | 0.12 | **0.04** | **0.05** | **0.04** | **0.04** | **0.05** | 0.41 | 0.29 |

**Table 5: Average C2D2 results for small real-world graphs**

| | FF | FS | IRE | MHRW | RD | RDN | RE | RJ | RN | RNE | SB | XS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Gnutella | 0.04 | **0.05** | **0.05** | 0.19 | 0.14 | **0.05** | **0.05** | **0.04** | **0.01** | **0.02** | **0.1** | **0.07** |
| HepPh | 0.16 | 0.21 | 0.21 | 0.18 | 0.39 | 0.22 | 0.21 | 0.17 | **0.01** | **0.09** | 0.31 | 0.43 |
| HepTh | 0.17 | 0.18 | 0.17 | 0.13 | 0.32 | 0.18 | 0.17 | 0.13 | **0.01** | **0.06** | 0.33 | 0.33 |
| Cora | 0.18 | 0.21 | 0.21 | 0.28 | 0.42 | 0.22 | 0.21 | 0.17 | **0.01** | **0.09** | 0.27 | 0.37 |
| Caida | 0.30 | 0.28 | 0.24 | 0.33 | 0.47 | 0.26 | 0.24 | 0.18 | **0.00** | **0.06** | 0.38 | 0.33 |
| Topology | 0.21 | 0.21 | 0.23 | 0.40 | 0.43 | 0.3 | 0.23 | 0.18 | **0.01** | 0.12 | 0.34 | 0.34 |

**Table 6: Average C2D2 results for large real-world graphs**

observed that high path lengths in Road and Bus graphs result in poor HPD2 results for most algorithms.

| | FF | XS | RJ | FS | MHRW | RN | RNE | RE | IRE | RDN | SB | RD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Email | 0.37 | 0.55 | 0.27 | 0.80 | 0.15 | 0.49 | 0.79 | 0.49 | 0.31 | 0.31 | 0.53 | 0.18 |
| Bio | 0.31 | 0.18 | 0.20 | 0.46 | 0.31 | 0.56 | 0.87 | 0.43 | 0.19 | 0.26 | 0.12 | 0.29 |
| Bus | 0.35 | 0.15 | 0.56 | 0.42 | 0.75 | 0.96 | 0.99 | 0.85 | 0.71 | 0.89 | 0.89 | 0.92 |
| Road | 0.43 | 0.21 | 0.63 | 0.44 | 0.66 | 0.97 | 0.99 | 0.89 | 0.80 | 0.94 | 0.90 | 0.94 |
| Wiki | 0.30 | 0.35 | 0.19 | 0.42 | 0.17 | 0.62 | 0.91 | 0.39 | 0.20 | 0.27 | **0.07** | 0.39 |
| ISP | 0.36 | 0.46 | 0.23 | 0.52 | 0.27 | 0.63 | 0.95 | 0.54 | 0.29 | 0.37 | 0.14 | 0.48 |

**Table 7: Average HPD2 results for small real-world graphs**

| | FF | FS | IRE | MHRW | RD | RDN | RE | RJ | RN | RNE | SB | XS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HepPh | 0.24 | 0.93 | **0.10** | 0.63 | 0.16 | **0.10** | 0.91 | **0.09** | 0.53 | 0.97 | 0.55 | 0.28 |
| HepTh | 0.29 | 0.82 | 0.29 | 0.52 | 0.43 | 0.34 | 0.83 | 0.11 | 0.33 | 0.94 | 0.32 | 0.41 |
| Cora | 0.25 | 0.78 | 0.11 | 0.62 | 0.19 | 0.11 | 0.86 | 0.14 | 0.47 | 0.98 | 0.15 | 0.11 |
| Topology | 0.17 | 0.33 | 0.25 | 0.68 | 0.28 | 0.38 | 0.56 | 0.20 | 0.21 | 0.94 | 0.33 | 0.23 |

**Table 8: Average HPD2 results for large real-world graphs.**

*Large graphs.* Table 8 represents HPD2 results for four large real graphs. This table and Table 9 indicate that on average RJ and IRE can better preserve distances (RJ was also good in small graphs). RJ, RDN, and IRE result in low HPD2 for the HepPh with a high $H(deg)$ and low $EBC$ relevant to these algorithms. We observe that most algorithms have lower HPD2 for large graphs. These graphs have lower diameters (Table 2) or path-related features, which is important for most algorithms (Figure 2(c)). Therefore, $H(deg)$, $EBC$ and path-related features appear to be important for the HPD2.

*Overall results.* The average results of three metrics in Table 9 indicate sampling algorithms' quality regarding scale and type of graphs (for HPD2 metric we only consider four large real graph

| | D3 | | | C2D2 | | | HPD2 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Syn | Real | | Syn | Real | | Syn | Real | |
| | | Small | Large | | Small | Large | | Small | Large |
| FF | 0.61 | 0.15 | 0.20 | 0.18 | 0.23 | 0.18 | 0.20 | 0.35 | 0.24 |
| FS | 0.77 | 0.33 | 0.39 | 0.15 | 0.17 | 0.19 | 0.69 | 0.51 | 0.71 |
| IRE | 0.71 | 0.27 | 0.16 | 0.15 | 0.13 | 0.18 | 0.28 | 0.42 | 0.19 |
| MHRW | 0.66 | 0.22 | 0.40 | 0.17 | 0.30 | 0.25 | 0.31 | 0.38 | 0.61 |
| RD | 0.65 | 0.38 | 0.17 | 0.20 | 0.42 | 0.36 | 0.44 | 0.56 | 0.27 |
| RDN | 0.79 | 0.34 | 0.19 | 0.16 | 0.14 | 0.21 | 0.34 | 0.51 | 0.23 |
| RE | 0.89 | 0.65 | 0.63 | 0.15 | 0.14 | 0.18 | 0.66 | 0.78 | 0.79 |
| RJ | 0.65 | 0.17 | 0.15 | 0.15 | 0.15 | 0.15 | 0.25 | 0.35 | 0.13 |
| RN | 0.87 | 0.59 | 0.57 | **0.09** | **0.06** | **0.01** | 0.43 | 0.49 | 0.38 |
| RNE | 0.90 | 0.74 | 0.72 | 0.12 | **0.10** | **0.07** | 0.78 | 0.92 | 0.96 |
| SB | 0.79 | 0.43 | 0.40 | 0.23 | 0.25 | 0.29 | 0.60 | 0.44 | 0.34 |
| XS | 0.60 | 0.27 | 0.25 | 0.36 | 0.23 | 0.31 | 0.30 | 0.32 | 0.26 |

**Table 9: Average results for different graph categories**

results). The algorithms can better preserve degree distribution for real graphs and many algorithms have better sampling quality for large real graphs. However, regarding CC most algorithms have better sampling quality for synthetic graphs and RN and RNE perform better on large real graphs. Regarding HPD2 most algorithms have better results on large real graphs, due to the lower diameters.

## 7 CONCLUSION AND FUTURE WORK

We investigated the quality of samples by twelve sampling algorithms of node, edge, and traversal-based categories under D3, C2D2, and HPD2 metrics. We evaluated them using several synthetic graphs of six types and twelve small and large real graphs. Our experiments show different characteristics of algorithms. XS and RJ better capture the degree distribution of synthetic and real graphs respectively. RN results in better samples regarding CC for all graph types. RJ produces better samples regarding hop-plots. Correlation analysis and verification on large real graphs represented the impact of *EIC* (usually high in citation or social networks), path-related features and $CC_{var}$ on D3 results of most algorithms. While, $|N|$ and *ConCS* are relevant to C2D2. $H(deg)$, *EBC* and path-related features are most correlated with HPD2 results. We also discovered inconsistent patterns in large graphs compared with small graphs. As a particular result, the correlation analysis revealed no significant dependency on the sampling rate. Overall, we observed better sample quality of most algorithms on large real graphs under D3 and HPD2 metrics, which is promising for large-scale scenarios.

This work is beneficial to selecting an appropriate sampling algorithm regarding the desired topological property of samples having graph features. It can guide researchers in developing sampling quality predictors by selecting the most relevant features. It can also have implications for understanding algorithms and provide better estimations for original graph properties by considering the most correlated features.

We will conduct more experiments in the future, including larger synthetic and real graphs, other sampling quality metrics, and more sampling algorithms. Furthermore, we will analyze the results using other methods, such as mutual information.

## ACKNOWLEDGMENTS

Graph-Massivizer receives funding from the Horizon Europe research and innovation program of the European Union. Its grant management number is 101093202:https://graph-massivizer.eu/.

## REFERENCES

[1] Sami Abu-El-Haija et al. 2023. SubMix: learning to mix graph sampling heuristics. In *Uncertainty in Artificial Intelligence*. PMLR.
[2] Nesreen K Ahmed et al. 2013. Network sampling: From static to streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 8 (2013), 1–56.
[3] Simon Geisler et al. 2021. Robustness of graph neural networks at scale. *Advances in Neural Information Processing Systems* 34 (2021), 7637–7649.
[4] Minas Gjoka et al. 2010. Walking in facebook: A case study of unbiased sampling of osns. In *2010 Proceedings IEEE Infocom*. Ieee, 1–9.
[5] Leo A Goodman. 1961. Snowball sampling. *The annals of mathematical statistics* (1961), 148–170.
[6] Pili Hu and Wing Cheong Lau. 2013. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865* (2013).
[7] Johannes Illenberger and Gunnar Flötteröd. 2012. Estimating network properties from snowball sampled data. *Social Networks* 34, 4 (2012), 701–711.
[8] Kshitijaa Jaglan et al. 2023. Tight Sampling in Unbounded Networks. *arXiv preprint arXiv:2310.02859* (2023).
[9] Sang Hoon Lee et al. 2006. Statistical properties of sampled networks. *Physical review E* 73, 1 (2006), 016102.
[10] Jure Leskovec et al. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 177–187.
[11] Jure Leskovec et al. 2007. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 2–es.
[12] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 631–636.
[13] Arun S Maiya and Tanya Y Berger-Wolf. 2010. Sampling community structure. In *Proceedings of the 19th international conference on World wide web*. 701–710.
[14] Anna Myakushina. [n. d.]. Exploring Sampling Techniques in Large Graphs and Networks. ([n. d.]).
[15] Christopher R Palmer et al. 2002. ANF: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 81–90.
[16] Karl Pearson. 1896. VII. Mathematical contributions to the theory of evolution.—III. Regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* 187 (1896), 253–318.
[17] Bruno Ribeiro and Don Towsley. 2010. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. Association for Computing Machinery, 390–403. https://doi.org/10.1145/1879141.1879192
[18] Bruno Ribeiro and Don Towsley. 2012. On the estimation accuracy of degree distributions from graph sampling. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 5240–5247.
[19] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. https://networkrepository.com
[20] Benedek Rozemberczki et al. 2020. Karate Club: an API oriented open-source python framework for unsupervised learning on graphs. In *Proceedings of the 29th ACM international conference on information & knowledge management*. 3125–3132.
[21] Daniel A Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. 81–90.
[22] Michael PH Stumpf et al. 2005. Subnets of scale-free networks are not scale-free: sampling properties of networks. *Proceedings of the National Academy of Sciences* 102, 12 (2005), 4221–4224.
[23] E Upfal and M Mitzenmacher. 2005. Probability and computing.
[24] Elli Voudigari et al. 2016. Rank degree: An efficient algorithm for graph sampling. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 120–129.
[25] Yanhong Wu et al. 2016. Evaluation of graph sampling: A visualization perspective. *IEEE transactions on visualization and computer graphics* (2016).
[26] Sooyeon Yoon et al. 2007. Statistical properties of sampled networks by random walks. *Physical Review E* 75, 4 (2007), 046114.
[27] Muhammad Irfan Yousuf et al. 2023. Empirical characterization of graph sampling algorithms. *Social Network Analysis and Mining* 13, 1 (2023), 66.
[28] Muhammad Irfan Yousuf and Suhyun Kim. 2018. List sampling for large graphs. *Intelligent Data Analysis* 22, 2 (2018), 261–295.
[29] Fangyan Zhang et al. 2015. A visual and statistical benchmark for graph sampling methods. In *Exploring Graphs at Scale Workshop*, Vol. 3.

# Building Massive Knowledge Graphs using an Automated ETL Pipeline

Aaron Eberhart
metaphacts GmbH
Germany
ae@metaphacts.com

Peter Haase
metaphacts GmbH
Germany
ph@metaphacts.com

Wolfgang Schell
metaphacts GmbH
Germany
ws@metaphacts.com

## ABSTRACT

Knowledge graphs are extremely versatile semantic tools, but there are current bottlenecks with expanding them to a massive scale. This concern is a focus of the Graph-Massivizer project, where solutions for scalable massive graph processing are investigated. In this paper we'll describe how to build a massive knowledge graph from existing information or external sources in a repeatable and scalable manner. We go through the process step-by-step, and discuss how the Graph-Massivizer project supports the development of large knowledge graphs and the considerations necessary for replication.

## CCS CONCEPTS

• **Information systems → Data exchange**; **Mediators and data integration**.

## KEYWORDS

Graph-Massivizer; metaphactory; ETL; RDF

**ACM Reference Format:**
Aaron Eberhart, Peter Haase, and Wolfgang Schell. 2024. Building Massive Knowledge Graphs using an Automated ETL Pipeline. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '24 ), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3652900

## 1 INTRODUCTION

A knowledge graph is a flexible semantic tool that can serve as the foundation for a wide variety of information representation purposes and use cases, such as fast-tracking drug discovery and reducing research costs, smart manufacturing solutions to support human manufacturing planners, and global fraud detection and risk management. It also unlocks AI initiatives by enriching existing black-box solutions with machine-interpretable semantics and adding a layer of trust and transparency. While knowledge graphs are extremely versatile, there are current bottlenecks with expanding them to a massive scale. This concern is a focus of the Graph-Massivizer [3] project, where solutions for scalable massive graph processing are investigated. In this paper we'll describe how to build a massive knowledge graph from existing information or external sources in a repeatable and scalable manner. We go through the process step-by-step, and discuss how the Graph-Massivizer project supports the development of multiple large knowledge graphs and the considerations necessary for replication.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE Companion '24 , May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3652900

### 1.1 Knowledge Graphs

Knowledge graphs are large networks of entities representing real-world objects, like people and organizations, and abstract concepts, like professions and topics, and their semantic relations and attributes. Knowledge graphs help organizations centralize, organize and understand internal data, often stored away in disparate sources. Depending on the volume of data a knowledge graph varies in size, ranging from a simple knowledge graph of a few to one with an extensive repository with millions of entities and interlinked relations.

There are multiple approaches to creating a knowledge graph this large in size, including using an ETL (extract-transform-load) or ELT (extract-load-transform) pipeline, which we'll explore in this paper. The ETL pipeline was developed as part of Graph-Massivizer, an EU-funded research project dedicated to researching and developing a scalable, sustainable and high-performing platform based on the massive graph representation of extreme data.

### 1.2 The Graph-Massivizer Project

The Graph-Massivizer project is developing a suite of five open-source software tools encompassing the sustainable life cycle of processing extreme data as massive graphs. These massive graphs support use cases such as green AI for a sustainable automotive industry, a data center digital twin for sustainable exascale computing and more.

The tools focus on holistic usability (from extreme data ingestion and massive graph creation), automated intelligence (through analytics and reasoning), performance modeling, and environmental sustainability tradeoffs, supported by credible data-driven evidence across the computing continuum. For example, the Graph-Massivizer's Graph-Inceptor tool is designed for creating knowledge graphs and storing graph data, offering two primary services:

(1) An extract, transform and load (ETL) pipeline requiring deployment to an IT cloud infrastructure consisting of servers, storage systems and databases
(2) A graph processing framework delivered as a Java library and made available in HPC clusters

Furthermore, the project consortium aims to create an integrated platform that is user-friendly and easy to deploy in enterprise environments The platform will tightly integrate the tools developed by Graph-Massivizer to provide a comprehensive offering.

## 2 KNOWLEDGE GRAPH CREATION PROCESS

Once a user has decided on their approach, they can start creating their knowledge graph, which involves many different tasks and phases. We'll explore in-depth some of the aspects to consider for the knowledge graph creation process.

### 2.1 FAIR Data Principles

A knowledge graph is only one system within an enterprise environment consisting of interconnected systems and data sources. One key aspect in this world of interconnected systems and data is following the FAIR [5] data principles, which ensures the reusability and interoperability of a knowledge graph, allowing users to enrich and extend their knowledge graph for various use cases and applications.

A knowledge graph supports the FAIR principles with the use of unique and persistent identifiers for entities, providing linked metadata describing the origin and modalities for accessing and using datasets, the use of semantic data models, and building on open standards for storing, accessing, and querying data. The following sections provide more information on how this works in detail.

### 2.2 Graph data model

Graphs are represented using the Resource Description Framework (RDF) [6], a semantic web standard for data interchange on the web. By using RDF, a graph can be expressed as a set of statements (or triples), each of which describes a single fact It allows for easy merging, linking and sharing of structured and semi-structured data across various systems and applications.

In this paper we only consider RDF-based graphs. There are other graph models, e.g.Labeled Property Graphs (LPG). Using RDF-star[1], any graph can be expressed, so RDF-star can also be used as a bridge to and from Labeled Property Graphs. RDF-star is an extension of RDF and also supports expressing statements on statements, which allows one to model edges with attributes.

### 2.3 Iterative approach

When creating a knowledge graph from scratch, it is useful to apply an iterative approach, involving:

(1) identifying source datasets and making them accessible
(2) defining a semantic data model using ontologies and vocabularies
(3) defining RDF mappings to convert from structured source data to RDF
(4) pre-processing source data (per file), e.g., to clean up data
(5) performing RDF conversion using the provided mappings
(6) post-processing intermediate results (per file), e.g., to create additional relations or aggregate data
(7) loading RDF data into the knowledge graph to persist the data in a graph database
(8) post-process intermediate results (whole graph), e.g., to create additional relations or aggregate data
(9) performing data validation to ensure the graph conforms to the defined data model.

[1]https://www.w3.org/groups/wg/rdf-star/publications/


**Figure 1: Iterative KG Creation Approach**

When violations are observed during data validation, the results can be used as a starting point to improve the pipeline. For example, source data can be fixed by performing data cleansing, adjusting the ontology or RDF mappings, or performing another iteration of the data integration process or ETL pipeline.

### 2.4 Providing dataset metadata

Data catalogs are a core building block for any FAIR data implementation, as they connect the available data assets with the knowledge graph. They support both interoperability as well as accessibility, as defined in the FAIR data principles.

In this approach, the data catalog is represented as a knowledge graph itself. It is semantically described with descriptive metadata and access metadata and is interlinked with other parts of the knowledge graph—such as ontologies and vocabularies—and it is embedded into and connected with data assets. Dataset descriptions (or data catalogs) are based on open and extensible W3C standards (e.g., DCAT) to make the data discoverable, accessible and traceable. With dataset descriptions, humans and machines (i.e., AI/ML algorithms) can consume data in context since the data is directly linked to the models and dataset descriptions, which themselves are based on open standards, are shareable and can even be queried all at once through a single, semantic query language.

### 2.5 Semantic Data Model

The next step in creating the knowledge graph is defining the data model. A knowledge graph typically follows one or multiple well-defined schemas which are specified using ontologies and vocabularies.

*2.5.1 Ontologies.* Ontologies are semantic data models that define the types of entities that exist in a domain and the properties that can be used to describe them. An ontology combines a representation, formal naming and definition of the elements (such as classes, attributes and relations) that define the domain of discourse. One may think of it as the logical graph model that defines what types (sets) of entities exist, their shared attributes and logical relations.

Ontologies can be specified using open standards like Web Ontology Language (OWL) [1] and Shapes Constraint Language (SHACL) [2].

*2.5.2 Vocabularies.* Vocabularies are controlled term collections organized in concept schemes that support knowledge graph experts, domain experts and business users in capturing business-relevant terminology. A term could include preferred and alternative labels (synonyms) in multiple languages and carries natural language definitions. Terms can be related to each other or defined as loosely related. The most common examples of different types of vocabularies are thesauri, taxonomies, terminologies, glossaries, classification schemes and subject headings, which can be managed using SKOS as an open standard.

## 2.6 RDF Mappings

The mapping process enables simple conversion, from a huge amount of source data to RDF, in an automated fashion. Converting structure data to RDF can be done by mapping certain elements and attributes from the source files to RDF data using a set of mapping rules.

As an example, all values of a column in a CSV file or a table in a relational database are mapped to RDF statements with the row's unique key being mapped to a subject IRI, the column to a predicate and the row value to the object position of a triple. Mapping rules can be provided either in a declarative way or programmatically.

*2.6.1 Declarative mappings.* Declarative mappings follow the no-code approach, meaning they can be defined using a simple text editor or visual tools, without requiring special programming skills.

The mappings are defined using the standardized Relational Mapping Language (RML). RML itself is also based on RDF, so both data model (ontology), mappings (RML maps) and instance data all use the same format. RML supports both tabular/relational and hierarchical data structures in formats like CSV, JSON or XML. Support for other formats can be provided as well.

RML defines just the mapping language. A wide range of implementations in the form of mapping engines (most of them open-source) are available. They can be used either as stand-alone tools or embedded into custom applications as a library.

*2.6.2 Programmatic mappings.* Implementing the mapping process using a custom program is the most flexible way to convert data to RDF. All means provided by the programming language and its ecosystem— such as frameworks and libraries—can be used (e.g., accessing data in various formats). Also, language-specific connectors, such as JDBC to access relational databases in the Java programming language, or web service connectors provide great flexibility. The biggest advantage is full control over the mapping process, as any kind of algorithm, data generation, use of caches and memory, navigating data structure or control flow is possible.

*2.6.3 Choosing between declarative or programmatic approach.* Using declarative mappings based on RML is the quickest and easiest way to implement mappings from structured data to RDF, as it follows a pre-defined approach that covers many use cases and formats and does not require special programming skills.

Only when declarative mappings do not suffice for the mapping at hand, should mappings be implemented as a custom program. While programmatic mappings allow for greater flexibility, this approach also requires more effort and programmatic skills, which are not necessarily available to people implementing a data pipeline.

In some cases where declarative mappings support most data structures to be mapped to RDF and only a few more complicated cases cannot be covered, a hybrid approach may be suitable. In that case, most mappings would be implemented declaratively in RDF and only a few special cases be handled by custom coding.

## 2.7 Performing pre- and post-processing

Besides converting source data as-is to RDF, sometimes additional steps are required to conform to the graph data model. This may be performed as pre- or post-processing steps, either on the original source before the RDF conversion or after.

Pre-processing steps typically work on the unit of a single source file. Typical examples are data cleansing, filtering of invalid data, splitting out units from numerical values, and datatype conversions to conform with certain numeric or date-time formats.

Post-processing steps may either be performed on the intermediate RDF files or the whole graph. Typical examples are tasks such as: specify the named graph for a set of statements, update graph metadata, such as the timestamp of last update of a dataset based on source data, and others.

## 2.8 Data Ingestion

The result of the previous steps is composed of a set of files in RDF format. This set of files may already be used to distribute the data in RDF format, e.g. as a data product.

As a next step, ingesting this file-based dataset into a graph database provides a base for easy querying and graph analytics supported by the database engine.

In addition to loading the data into the database for querying using the SPARQL query language, creating a full-text search index enables additional capabilities when searching for textual data in the graph. This is typically handed off from the database to specialized and tightly integrated full-text search engines like Lucene[2], Solr[3], or Elasticsearch[4].

## 2.9 Performing data validation

Once all data has been converted to RDF and is ingested in the database, it can be submitted to a data validation to ensure good data quality.

When defining the ontology using OWL and SHACL, the model description can be used to automatically validate the database and ensure that data follows the defined model. This can be done using a so-called SHACL engine, which verifies that the data in the database adheres to the shapes defined in the ontology. SHACL engines are provided by (commercial) RDF databases as well as open-source projects or commercial tools such as metaphactory.

[2]https://lucene.apache.org/
[3]https://solr.apache.org/
[4]https://www.elastic.co/

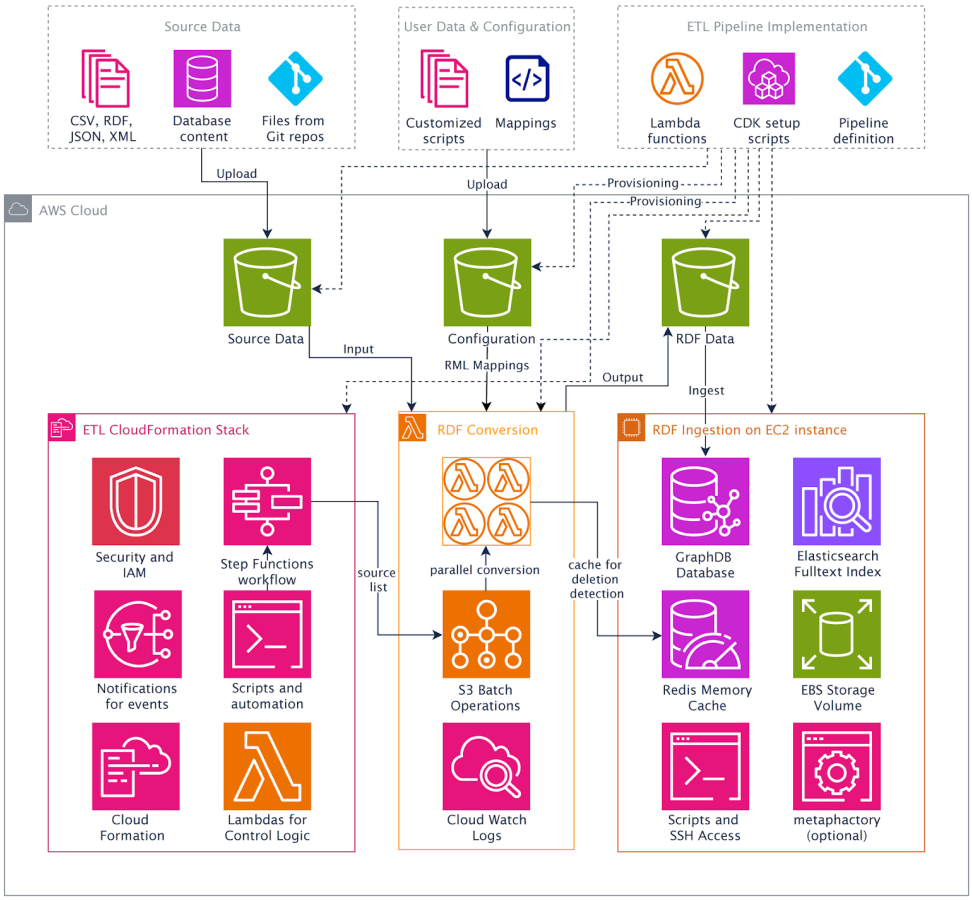



**Figure 2: ETL Architecture**

## 3 ARCHITECTURE

The pipeline uses a multitude of AWS services to implement the RDF conversion and ingestion process with a cloud-native approach resulting in high parallelization and efficient use of resources. Details on this architecture can be found in Figure 2.

## 4 EXAMPLE

The Graph Massivizer use cases from the data center, industrial, and financial domains are based on either commercial, internal or sensitive datasets, so they cannot be used for a public demonstration. Instead, we'll use the Dimensions Covid Dataset[5] [4] as an example of a large, publicly available dataset to create a scientific knowledge graph using the ETL pipeline.

The dataset provides information on global publications, academic papers, authors, research organizations, funders, grants, datasets and clinical trials. The zipped dataset (1.09GB) is available for download on Figshare. The data files are in CSV format, the fields are described in the documentation of the main Dimensions dataset (although not all documented fields are available in this publicly available subset).

The semantic data model and dataset description as well as the corresponding RML mappings are provided as an example in the ETL pipeline Git repository[6].

## 5 FUTURE WORK

The ETL pipeline will be extended and integrated to work with the Graph-Massivizer toolkit. This process will continue through the duration of the project, adapting to evolving project needs and use cases.

In parallel with Graph-Massivizer developments, work on the ETL pipeline will also provide and extend capabilities with the metaphactory platform.

**Acknowledgement** This project has received funding from the European Union's Horizon Research and Innovation Actions under Grant Agreement N° 101093202.[7]

[5] https://www.dimensions.ai/covid19/

[6] https://github.com/metaphacts/metaphacts-etl-pipeline

[7] More information available at: https://graph-massivizer.eu/

# REFERENCES


[1] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah McGuinness, Peter Patel-Schneijder, and Lynn Andrea Stein. 2004. *OWL Web Ontology Language Reference*. Recommendation. World Wide Web Consortium (W3C). See http://www.w3.org/TR/owl-ref/.

[2] Dimitris Kontokostas and Holger Knublauch. 2017. *Shapes Constraint Language (SHACL)*. W3C Recommendation. W3C. https://www.w3.org/TR/2017/REC-shacl-20170720/.

[3] Radu Prodan, Dragi Kimovski, Andrea Bartolini, Michael Cochez, Alexandru Iosup, Evgeny Kharlamov, Jože Rožanec, Laurenţiu Vasiliu, and Ana Lucia Vărbănescu. 2022. Towards Extreme and Sustainable Graph Processing for Urgent Societal Challenges in Europe. In *2022 IEEE Cloud Summit*. 23–30. https://doi.org/10.1109/CloudSummit54781.2022.00010

[4] Dimensions Resources. 2021. Dimensions COVID-19 publications, datasets and clinical trials. (9 2021). https://doi.org/10.6084/m9.figshare.11961063.v42

[5] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3 (2016).

[6] David Wood, Markus Lanthaler, and Richard Cyganiak. 2014. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. W3C. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

# Serverless Workflow Management on the Computing Continuum: A Mini-Survey

Reza Farahani
reza.farahani@aau.at
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria

Frank Loh
frank.loh@uni-wuerzburg.de
University of Würzburg
Würzburg, Germany

Dumitru Roman
dumitru.roman@sintef.no
Sintef
Oslo, Norway

Radu Prodan
radu.prodan@aau.at
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria

## ABSTRACT

The growing desire among application providers for a cost model based on *pay-per-use*, combined with the need for a seamlessly integrated platform to manage the complex workflows of their applications, has spurred the emergence of a promising computing paradigm known as *serverless* computing. Although serverless computing was initially considered for cloud environments, it has recently been extended to other layers of the *computing continuum*, i.e., edge and fog. This extension emphasizes that the proximity of computational resources to data sources can further reduce costs and improve performance and energy efficiency. However, orchestrating the computing continuum in complex application workflows, including a set of serverless functions, introduces new challenges. This paper investigates the opportunities and challenges introduced by serverless computing for *workflow management systems* (WMS) on the computing continuum. In addition, the paper provides a taxonomy of state-of-the-art WMSs and reviews their capabilities.

## CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

## KEYWORDS

Workflow; Workflow Management Systems (WMS); Serverless Computing; Function-as-a-Service (FaaS); Edge-Cloud Continuum; Function Scheduling; Service Orchestration; Sustainability.

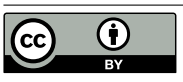
**ACM Reference Format:**
Reza Farahani, Frank Loh, Dumitru Roman, and Radu Prodan. 2024. Serverless Workflow Management on the Computing Continuum: A Mini-Survey. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3652901

## 1 INTRODUCTION

The proliferation of online applications, advancements in networking and computing technologies, and the continuously growing number of users who opt for diverse online services have collectively propelled application *workflow management systems* (WMS) to the forefront of discussions among various stakeholders [54]. In this context, an application workflow refers to cooperative tasks, activities, or processes that execute a specific business or computational logic. These tasks require network, computational, and storage resources beyond the capabilities of a single on-premises cluster. The emergence of cloud computing has revolutionized the domain of WMSs by offering scalable resources and diverse services. Most public cloud providers offer WMS, such as Amazon Simple Workflow Service and Google Cloud Composer, enabling application providers to build, deploy, schedule, and orchestrate their workflow tasks comprehensively. Although the adoption of cloud services represented a significant advancement, challenges persist in realizing a pure pay-per-use model and achieving seamless scalability. This is because cloud providers typically charge application owners based on allocated resources rather than actual consumption. Furthermore, application providers are burdened with ongoing responsibilities to configure and scale infrastructure instances, requiring comprehensive application monitoring and expertise in both infrastructure and services management [49].

To address the described challenges, both application and cloud providers made substantial architectural modifications. On the application side, the architecture transitioned from monolithic to service-oriented, then to microservices [14], and *Function-as-a-Service* (FaaS) [8], allowing the execution of small pieces of code as functions [46]. Taking into account the distinctive characteristics of emerging applications, particularly those with FaaS-based architectures, cloud providers have taken advantage of their previous experience, e.g., virtualization and containerization paradigms, to establish a pure pay-per-use paradigm known as *serverless* [53] as an alternative to the Infrastructure-as-a-Service (IaaS) model. Hence, cloud providers are increasingly adopting serverless principles across a spectrum of their existing services, encompassing serverless containers (e.g., AWS Fargate or Google Cloud Run), serverless databases (e.g., AWS DynamoDB), and even serverless graph processing (e.g., AWS Neptune). Furthermore, most cloud providers offer serverless WMSs, such as AWS Step Functions, Google Workflows, and IBM Composer.

In pursuit of cost efficiency, reduced latency, and energy conservation, both industry and academia have recently increased their



This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, UK*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652901

efforts to utilize fog or edge resources in close proximity to application users, thus, establishing serverless WMSs throughout the entire *computing continuum* [20]. To this end, open-source serverless platforms, e.g., Apache OpenWhisk [1] or OpenFaaS [38], have been developed to operate on on-promise or leased edge or fog instances. However, this presents a challenge, given that the serverless paradigm was originally designed for cloud environments, not accounting for such computational constraint instances. Hence, among numerous critical considerations, one key research question in designing serverless WMSs on the computing continuum is: "How can *service level objectives* (SLOs), e.g., latency and energy, and economic cost, be optimized by determining the strategic allocation of workflow functions, deciding which to execute on edge or fog instances and which to offload to the public cloud?" This mini-survey explores fundamental concepts, reviews the latest WMSs, and discusses the opportunities and challenges arising from integrating serverless paradigms into WMSs.

## 2 BACKGROUND

### 2.1 Computing Continuum

The computing continuum is a structural design consisting of various computing and storage resources with varying network bandwidth, interconnected in three layers: Cloud, Fog, and Edge [18, 23]. The *cloud layer* is supported by public providers, such as AWS, IBM, or Google, to provide large-scale infrastructure and a broad range of services. The *fog layer* presents computing capabilities in close proximity to data sources and user devices on a smaller scale. This involves utilizing less powerful devices with lower access latencies compared to cloud servers, typically located in network base stations (e.g., gNodeB in 5G). The *edge layer* consists of local servers and devices with limited resources, including sensors and actuators, equipped with computational capabilities. These are strategically placed at user locations to further minimize service latency. Although each layer can collaborate and exchange information in the execution of applications, it also possesses the ability to operate independently. Adopting this multi-layer architecture empowers application providers to utilize appropriate resources and services, consequently enhancing SLOs, energy, and economic cost compared to dependence on a singular layer of resources.

### 2.2 Serverless Computing

Serverless is an emerging computing paradigm designed for the deployment of FaaS applications and other services statelessly [3]. Serverless applications typically comprise a collection of stateless and atomic functions, commonly deployed within containers or encapsulated as Zip files. Upon invocation of the function by the application, a *cold start* occurs on the infrastructure side, requiring the deployment of the container from online repositories to the specified resource. In contrast, a *warm start* occurs when the requested function is pre-deployed on the computational resource, resulting in rapid initialization and execution. In such paradigms, functions interact and exchange data using platform services like databases, if necessary [28]. Therefore, it empowers application developers to build scalable and event-driven applications while incurring charges based on the execution time of functions (i.e., pay-per-use), in addition to any supplementary services utilized by these functions. Moreover, it eliminates the complexities associated with the provisioning and maintenance of resources, challenges commonly encountered in traditional cloud-based systems designed for monolithic applications [49]. For a comprehensive overview of the serverless lifecycle, we refer to literature [37].

### 2.3 Workflow Management Systems

Application workflows typically consist of multiple stages, each comprising a set of independent tasks. These workflows are commonly represented as *directed acyclic graphs* (DAGs), where nodes represent tasks and edges denote data dependencies. Task communication is generally based on shared file systems and task execution occurs only when all dependencies are satisfied [55]. A *workflow management system* (WMS) operates as a dedicated tool to execute and orchestrate such workflows in heterogeneous computing and storage resources, including local and cloud instances. Many WMSs, such as Pegasus [13], Airflow [22], Argo, AWS Step Functions, Google Workflows, or IBM Composer, plus open-source ones like LithOps [48], and PyWren [26], have been developed to facilitate the seamless execution and management of application workflows across diverse computing architectures with serverful or serverless models. Such WMSs typically receive an application DAG as input, actively monitor the progress of running tasks and available resources, and generate execution plans by mapping tasks to the existing computing resources to enforce rigorous adherence to data dependencies while simultaneously striving to minimize the overall execution time.

## 3 SERVERLESS WORKFLOW MANAGEMENT

The rising customer demands from major public cloud providers, such as AWS, Google, IBM, and Azure, for serverless platform integration, coupled with the increasing complexity of serverless workflows, have greatly boosted the popularity of serverless WMSs [12]. Statistics reveal a substantial six-fold increase in the adoption of serverless workflows in Azure between 2019 and 2022 [32]. Managing complex workflows, often involving multiple functions and adhering to specific SLO levels, presents a challenge that cannot be adequately addressed by a single cloud-based architecture with concurrent function execution limitations (e.g., 1,000 functions for AWS Lambda). Therefore, the prevalent adoption of open source serverless platforms such as OpenFaaS [38] or OpenWhisk [1] has become a common practice to equip the other two layers of the computing continuum with serverless capabilities. However, designing WMSs to meet requested SLO levels across various resources in the computing continuum while accounting for available resources and concurrency limitations presents a considerable and intricate challenge. In the following discussion, we categorize and review state-of-the-art WMSs into three distinct types.

### 3.1 Cloud-based WMSs

Currently, all leading cloud providers have established their proprietary serverless WMSs. For instance, AWS Step Functions, an Amazon serverless WMS, utilizes the JSON format to orchestrate workflow functions, employing various constructs for parallelization, data distribution, and conditional branches. Despite providing versatile constructs, both WMSs are limited in scalability as they

operate within the same cloud region of a single provider, preventing the use of other cloud regions in a federated cloud manner or the computing continuum. Google Workflows uses the YAML format to define control and data flows within workflows. Unlike the previously mentioned WMSs, it supports function execution through HTTP requests, allowing the deployment of serverless functions across any cloud region.

In recent years, various WMSs have been developed to execute workflows across single or multiple providers' regions. Spock [21] operates as a scalable and adaptive WMS, employing virtual machines and a serverless platform deployed in public clouds. Its objective is to distribute the execution of machine learning inference jobs to minimize SLO violations. Sequoya [51] is another that provides developers with multiple scheduling policies tailored to various Quality of Service (QoS) parameters. Once a function completes, it triggers successor functions either on the local server running OpenWhisk or on cloud servers. The Multi-Provider Serverless Computing (MPSC) framework [2] is one of the multi-cloud WMSs, aiming to optimize task allocation between local servers and cloud platforms, specifically AWS Lambda and IBM Cloud Functions. Hyperflow [36] is another WMS that enables the execution of workflow functions exclusively within a designated region of AWS Lambda or Google Cloud Functions. However, these WMSs mostly ignore the function concurrency limitations of cloud providers.

The literature has also introduced domain-specific serverless WMSs, designed to accelerate the development of serverless applications within specific domains. Examples include scientific workflows, which encompass complex and long-term data-intensive tasks [6, 25]. The mentioned WMSs use HyperFlow to construct WMSs and execute scientific workflows on AWS Lambda and Google Cloud Functions. Furthermore, numerous recent works aim to enhance the execution speed of workflows in public cloud environments [10, 30, 31, 33]. For this aim, Mahgoub et al. [31] introduced three levels of optimizations integrated on AWS Lambda, allocating the appropriate resources for each function invocation. They also introduced SONIC [30], which determines the optimal approach to pass data between various serverless functions, that is, local storage, direct passing, and remote storage.

### 3.2 Edge-Cloud Continuum-based WMSs

In the domain of edge-cloud WMSs, numerous works have focused mainly on specific tasks within WMSs, such as function scheduling [41, 50, 56, 58]. For instance, Aslanpour et al. proposes an energy-aware serverless scheduling method tailored for applications in edge computing in [4]. Two priority-based and zone-oriented algorithms improve the operational availability of bottleneck edge devices using "sticky offloading" and "warm scheduling" to optimize QoS metrics. Skippy [42] represents another container-based scheduling method within these types of WMS, strategically balancing trade-offs between data and computation exchange. It takes workload-specific compute requirements into account, including GPU acceleration, to optimize overall utilization. Numerous works, such as OSCAR [43], propose the offloading of functions to clouds when edge resources become overloaded. Skedulix is another system on the edge-cloud continuum [11] that offloads functions from OpenFaaS to AWS Lambda to minimize costs while adhering to deadline constraints. Similarly, Serverledge is a decentralized edge-cloud system [47] that runs serverless functions on edge devices and offloads them to cloud servers or neighboring edge instances in case of overload. However, such systems target small single-workflow scheduling due to the concurrency limitations of edge devices and a single cloud instance. Costless [17] is a framework designed to optimize the execution cost of single serverless workflow applications by dividing their execution between the edge layer and the cloud. However, it does not account for scheduling concurrent workflows. The authors of [29] investigated the placement of workflow functions in edge-cloud systems to only minimize completion time.

### 3.3 Simulations-based WMSs

Addressing the need for proactive performance evaluation and prediction in serverless WMSs, many dedicated simulators have recently been developed. These simulators not only aid in assessing performance or cost metrics before deployment and execution [24, 34, 45, 52], but strive to provide valuable predictions [5, 16] to serverless providers regarding diverse load and request patterns. The authors in [34] simulate serverless functions with a fixed memory setup in a single cloud region and model the average response time of the functions, cold start, and concurrent instances of the serverless function. SimLess [45] is another serverless simulator that assesses the overhead of individual functions, as well as the entire workflow, and their comparable setups in federated clouds. DFaaSCloud [24] as an extension of CloudSim [7] and OpenDC Serverless [27] are simulators specifically designed to simulate the execution time of workflow functions. Faas-sim [40] is an edge-cloud simulator offering a versatile serverless simulation environment backed by real-world trace data.

## 4 OPPORTUNITIES

### 4.1 Cost Model

Unforeseeable variations in application workloads pose a challenge when using fixed container provisioning, resulting in charges during inactive periods. While dynamic auto-scaling is an option for most IaaS providers, it introduces additional costs and the potential for imprecise resource provisioning. On the contrary, serverless computing charges are determined by actual triggered events, including dedicated resources and function invocation frequency. The serverless paradigm ensures more predictable pricing irrespective of workload fluctuations. By leveraging the precise scalability of serverless, avoiding unnecessary resource allocation and idle-time costs, the overall price remains unaffected by workload variability.

### 4.2 Scalability

Certain applications operate consistently in close proximity to data sources on the edge layer of the computing continuum, while several others require seamless integration throughout the computing continuum. Leveraging the execution of serverless functions on the computing continuum enables WMSs to carry out these functions optimally. For instance, many serverless WMSs operating within the computing continuum adopt a straightforward service execution strategy that involves task execution on edge instances as long as resources are available, with a subsequent transition to

offloading tasks to the fog or cloud when needed. Furthermore, concurrent function executions through parallelization techniques not only enhance the practicality but also boost the scalability of WMSs within the computing continuum.

### 4.3 Auto-scaling

Traditional IaaS infrastructure relying on virtual machines faced drawbacks such as large memory footprints and challenging scalability, involving duplication of significant data when creating more service replicas on an instance. Therefore, one of the critical challenges for IaaS-based resource-limited edge and fog layers lies in resource management. Embracing lightweight abstractions like containers, serverless solutions offer a smaller footprint and precise autoscaling. This efficiency is particularly notable because of the minimal overhead in creating or terminating replicas compared to full virtual machines. The promise is further enhanced when serverless adopts computation principles based on functions inherited from microservice architecture advancements instead of treating the entire application as a black box.

### 4.4 Statelessness

The statelessness feature of serverless architecture offers several advantages over a serverful architecture. Since each function or service operates independently without maintaining a persistent state between invocations, fault tolerance and resilience improved, since failures in one function do not impact the overall system. Moreover, it makes serverless platforms such as WMSs appealing for real-time collaboration tools such as instant messaging and chatbots [57]. Therefore, it enables WMSs to have more suitable performance, particularly for applications with inherent lack reliance on and awareness of previous event, compared to the serverful ones.

## 5 CHALLENGES

### 5.1 Stream Processing

While the serverless paradigm is widely acknowledged as a successful advancement for cloud computing, the scale-to-zero technique and the subsequent cold start of serverless functions may not be optimal for certain latency-sensitive stream processing applications [19, 37]. The ability to scale to zero has both advantages and drawbacks. Although it minimizes energy consumption and reduces economic costs, initial invocation of the function results in a cold start, introducing additional delays with current technologies. Although this delay poses no issue for batch applications, it can lead to performance degradation in stream time-sensitive processing applications, since they require real-time decision-making, particularly on resource-limited devices. This discrepancy has led to active exploration within the research community, with studies analyzing its impacts [15, 44] and proposing various promising solutions, such as reducing overhead in the development phase [53].

### 5.2 Data Distribution

Data and storage management are key in serverless WMSs, irrespective of the computational and network differences among computing continuum instances. The stateless nature of the serverless, coupled with a lack of server affinity, gives rise to challenges. While cloud-based WMSs uphold function states by storing them in storage, the proximity of maintaining this state becomes pivotal at the edge, consequently, the edge layer transmitting substantial data per function invocation to remote devices. Current serverless WMSs within the edge-cloud continuum tend to prioritize CPU and memory specifications while neglecting crucial storage provisioning techniques. Imagine a scenario where an application involves a sequence of function invocations. In this case, the transfer cost is incurred only once in traditional approaches for monolithic applications. However, with serverless, this cost may be incurred multiple times between any two consecutive function invocations if executed on different layers. In the cloud, these limitations are seamlessly addressed by robust data centers and a high-speed communication. In contrast, in the edge layer of the computing continuum, data transport becomes problematic [3, 35]. Although numerous research studies have been initiated that focus on data caching and storage placement have been initiated to address this problem [9, 39], we believe that more research is needed to fully address this challenge.

### 5.3 Complexity

The lack of effective abstractions for managing task-based workloads on serverless platforms, especially for workflows that exhibit intricate structures and dependencies, necessitates manual partitioning and encapsulation of workflow function codes. In addition, despite the overarching promise of serverless to diminish the need for manual resource management, it still requires the configuration and adjustment of resource-related parameters for application workflows, such as concurrency and memory per container. The mentioned requirements add complexity for the application providers. In addition, multiple layers of infrastructure within the computing continuum, coupled with middleware and execution engines, pose challenges in monitoring, understanding, and predicting application performance. Therefore, more research activities are needed to alleviate these complications.

## 6 CONCLUSION

This mini-survey explores relevant technologies and assesses the capabilities of serverless workflow management systems. We discuss the appropriateness of serverless WMSs on the computing continuum due to their (1) potential for seamless integration from cloud to edge, (2) provision of pure pay-per-use and cost-effective design, (3) mitigation of challenges related to resource provisioning and scaling, (4) facilitation of easy parallelization for stateless functions, and (5) provision of fine-grained scalability for resources. However, the design of such WMSs for today's applications faces significant challenges, including (1) prolonged latencies induced by cold startups, (2) difficulties in handling stream processing workflows, (3) complexities in managing data distribution and storage, (4) the intricate task of designing cost-, SLO-, and energy-aware systems, and (5) the potential to induce resource inefficiencies that necessitate collaborative efforts from both academia and industry.

## ACKNOWLEDGMENTS

Graph-Massivizer receives funding from the Horizon Europe research and innovation program of the European Union. Its grant management number is 101093202.

# REFERENCES


[1] 2023. OpenWhisk. https://openwhisk.apache.org/ Accessed: 2023-11-20.

[2] Austin Aske and Xinghui Zhao. 2018. Supporting Multi-Provider Serverless Computing on the Edge. In *Workshop Proceedings of the 47th International Conference on Parallel Processing (ICPP Workshops '18)*. ACM.

[3] Mohammad S Aslanpour et al. 2021. Serverless Edge Computing: Vision and Challenges. In *Proceedings of the 2021 Australasian Computer Science Week Multiconference*.

[4] Mohammad Sadegh Aslanpour et al. 2022. Energy-Aware Resource Scheduling for Serverless Edge Computing. In *2022 22nd IEEE Intl. Symp. on Cluster, Cloud and Internet Computing*. IEEE.

[5] Barcelona-Pons et al. 2021. Benchmarking Parallelism in FaaS platforms. *Future Generation Computer Systems* (2021).

[6] Krzysztof Burkat et al. 2021. Serverless Containers-Rising Viable Approach to Scientific Workflows. In *2021 IEEE 17th International Conference on eScience (eScience)*. IEEE.

[7] Rodrigo N Calheiros et al. 2011. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and experience* (2011).

[8] Paul Castro et al. 2019. The Rise of Serverless Computing. *Commun. ACM* (2019).

[9] Chen Chen et al. 2023. S-Cache: Function Caching for Serverless Edge Computing. In *Proceedings of the 6th International Workshop on Edge Systems, Analytics and Networking*.

[10] Anirban Das et al. 2020. Performance Optimization for Edge-Cloud Serverless Platforms via Dynamic Task Placement. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE.

[11] Anirban Das et al. 2020. Skedulix: Hybrid Cloud Scheduling for Cost-Efficient Execution of Serverless Applications. In *2020 IEEE 13th Intl. Conf. on Cloud Computing*. IEEE.

[12] Datadog. 2023. Datadog. The State of Serverless, August 2023. https://www.datadoghq.com/state-of-serverless/ Accessed: 2023-11-20.

[13] Ewa Deelman et al. 2015. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems* (2015).

[14] Paolo Di Francesco et al. 2019. Architecting with Microservices: A Systematic Mapping Study. *Journal of Systems and Software* (2019).

[15] Klimentina Djeparoska and Marjan Gusev. 2023. Limitations of AWS and GCP Serverless Functions. In *2023 31st Telecommunications Forum (TELFOR)*. IEEE.

[16] Simon Eismann et al. 2020. Predicting the Costs of Serverless Workflows. In *Int. Conf. on Performance Engineering*. ACM.

[17] Tarek Elgamal et al. 2018. Costless: Optimizing Cost of Serverless Computing through Function Fusion and Placement. In *2018 IEEE/ACM Symp. on Edge Computing*. IEEE.

[18] Reza Farahani et al. 2023. Towards Sustainable Serverless Processing of Massive Graphs on the Computing Continuum. In *Proc. of the 1st Workshop on Serverless, Extreme-Scale, and Sustainable Graph Processing Systems*.

[19] Marios Fragkoulis et al. 2023. A survey on the evolution of stream processing systems. *The VLDB Journal* (2023).

[20] Sukhpal Singh Gill et al. 2024. Modern computing: vision and challenges. *Telematics and Informatics Reports* (2024).

[21] Jashwant Raj Gunasekaran et al. 2019. Spock: Exploiting Serverless Functions for SLO and Cost Aware Resource Procurement in Public Cloud. In *2019 IEEE 12th Intl. Conf. on Cloud Computing*. IEEE.

[22] Scott Haines. 2022. Workflow Orchestration with Apache Airflow. In *Modern Data Engineering with Apache Spark: A Hands-On Guide for Building Mission-Critical Streaming Applications*. Springer.

[23] Matthijs Jansen et al. 2023. The SPEC-RG Reference Architecture for the Compute Continuum. In *2023 IEEE/ACM 23rd Intl. Symp. on Cluster, Cloud and Internet Computing*. IEEE.

[24] Hongseok Jeon et al. 2019. A CloudSim-Extension for Simulating Distributed Functions-as-a-Service. In *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*.

[25] Aji John et al. 2019. SWEEP: Accelerating Scientific Research Through Scalable Serverless Workflows. In *IEEE/ACM International Conference UCC Companion*. ACM.

[26] Eric Jonas et al. 2017. Occupy the Cloud: Distributed Computing for the 99%. In *Proceedings of the 2017 symposium on cloud computing*.

[27] S Jounaid. 2020. OpenDC Serverless: Design, Implementation and Evaluation of a FaaS Platform Simulator. Ph.D. thesis, Vrije Universiteit Amsterdam.

[28] Samuel Kounev et al. 2023. Serverless Computing: What It Is, and What It Is Not? *Commun. ACM* (2023).

[29] Liuyan Liu et al. 2019. Dependent Task Placement and Scheduling with Function Configuration in Edge computing. In *Proc. of the Intl. Symp. on Quality of Service*.

[30] Ashraf Mahgoub et al. 2021. {SONIC}: Application-aware Data Passing for Chained Serverless Applications. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 285–301.

[31] Ashraf Mahgoub et al. 2022. {ORION} and the Three Rights: Sizing, Bundling, and Prewarming for Serverless {DAGs}. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*.

[32] Ashraf Mahgoub et al. 2022. WISEFUSE: Workload Characterization and DAG Transformation for Serverless Workflows. *Proc. of the ACM on Measurement and Analysis of Computing Systems* (2022).

[33] Nima Mahmoudi et al. 2019. Optimizing Serverless Computing: Introducing an Adaptive Function Placement Algorithm. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*.

[34] Nima Mahmoudi and Hamzeh Khazaei. 2021. SimFaaS: A Performance Simulator for Serverless Computing Platforms. In *Int. Conf. on Cloud Computing and Services Science*.

[35] Redowan Mahmud et al. 2020. Application Management in Fog Computing Environments: A Taxonomy, Review and Future Directions. *ACM Computing Surveys (CSUR)* (2020).

[36] Maciej Malawski et al. 2020. Serverless Execution of Scientific Workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions. *Future Generation Computer Systems* (2020).

[37] Kien Nguyen et al. 2023. Serverless Computing Lifecycle Model for Edge Cloud Deployments. In *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE.

[38] OpenFaaS. 2023. OpenFaaS. https://www.openfaas.com/ Accessed: 2023-11-20.

[39] Li Pan et al. 2022. Retention-Aware Container Caching for Serverless Edge Computing. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE.

[40] Philipp Raith et al. 2023. faaS-sim: A Trace-Driven Simulation Framework for Serverless Edge Computing Platforms. *Software: Practice and Experience* (2023).

[41] Thomas Rausch et al. 2021. Optimized Container Scheduling for Data-Intensive Serverless Edge Computing. *Future Generation Computer Systems* (2021).

[42] Thomas Rausch et al. 2021. Optimized Container Scheduling for Data-Intensive Serverless Edge Computing. *Future Generation Computer Systems* (2021).

[43] Sebastián Risco et al. 2021. Serverless Workflows for Containerised Applications in the Cloud Continuum. *Journal of Grid Computing* (2021).

[44] Sashko Ristov et al. 2022. Colder than the Warm Start and Warmer than the Cold Start! Experience the Spawn Start in FaaS Providers. In *Proceedings of the 2022 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*.

[45] Sashko Ristov et al. 2022. SimLess: simulate serverless workflows and their twins and siblings in federated FaaS. In *Proceedings of the 13th Symposium on Cloud Computing*.

[46] Sashko Ristov et al. 2023. Large-scale Graph Processing and Simulation with Serverless Workflows in Federated FaaS. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*.

[47] Gabriele Russo Russo et al. 2023. Serverledge: Decentralized Function-as-a-Service for the Edge-Cloud Continuum. In *2023 IEEE Intl. Conf. on Pervasive Computing and Communications*. IEEE.

[48] Josep Sampe et al. 2021. Outsourcing Data Processing Jobs with Lithops. *IEEE Transactions on Cloud Computing* (2021).

[49] Hossein Shafiei et al. 2022. Serverless Computing: A survey of Opportunities, Challenges, and Applications. *Comput. Surveys* (2022).

[50] Yang Tang et al. 2020. Lambdata: Optimizing Serverless Computing by Making Data Intents Explicit. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*.

[51] Ali Tariq et al. 2020. Sequoia: Enabling Quality-of-Service in Serverless Computing. In *Proceedings of the 11th ACM symposium on cloud computing*.

[52] Erwin van Eyk. 2019. SimFaaS. https://github.com/erwinvaneyk/simfaas. Accessed: 2024-02-07.

[53] Erwin Van Eyk et al. 2018. Serverless Is More: From PaaS to Present Cloud Computing. *IEEE Internet Computing* (2018).

[54] Laurens Versluis and Alexandru Iosup. 2021. A survey of domains in workflow scheduling in computing infrastructures: Community and keyword analysis, emerging trends, and taxonomies. *Future generation computer systems* (2021).

[55] Fuhui Wu et al. 2015. Workflow scheduling in cloud: a survey. *The Journal of Supercomputing* (2015).

[56] Song Wu et al. 2021. Container Lifecycle-Aware Scheduling for Serverless Computing. *Software: Practice and Experience* (2021).

[57] Mengting Yan et al. 2016. Building a Chatbot with Serverless Computing. In *Proceedings of the 1st International Workshop on Mashups of Things and APIs*.

[58] Hanfei Yu et al. 2021. Harvesting Idle Resources in Serverless Computing via Reinforcement Learning. *arXiv preprint arXiv:2108.12717* (2021).

# Go-Network: a graph sampling library written in Go

Jože M. Rožanec
Jožef Stefan Institute
Ljubljana, Slovenia
joze.rozanec@ijs.si

Matias Rožanec
Facultad de Ingeniería, Universidad de Buenos Aires
Buenos Aires, Argentina
mrozanec@fi.uba.ar

## ABSTRACT

Go-Network is a Go language package for network generation and sampling. The core package provides basic data structures representing undirected graphs. Go-Network currently supports only integer values on graph nodes and edges. The library implements (a) data loading utilities supporting frequent graph formats, (b) algorithms for synthetic graph generation (e.g., Erdős-Rényi graphs), and thirty implementations of graph sampling algorithms. Among the many benefits the library inherits from Go (designed as a replacement for C++) are the compilation and execution speed (compiles directly to machine code) and its great support for concurrency while being memory savvy. These factors make the library a powerful tool for scientific purposes. We briefly describe the existing functionality, compare it against another graph sampling library (*Little Ball of Fur*), describe our design decisions, and draw attention to future work. Go-Network is publicly available and can be imported from https://github.com/graph-massivizer/go-network.

## CCS CONCEPTS

• **Software and its engineering → Software libraries and repositories**; • **Computing methodologies → Artificial intelligence**.

## KEYWORDS

Artificial Intelligence, Graph Sampling Algorithm, Scalable Graph Processing

**ACM Reference Format:**
Jože M. Rožanec and Matias Rožanec. 2024. Go-Network: a graph sampling library written in Go. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3652903

## 1 INTRODUCTION

Our environment is full of systems for which it is challenging to derive collective behavior based on the knowledge of its components. Such systems are known under the term *complex systems*. They can be modeled as networks to capture the interactions between the system's components of relevance to analyze the actual behavior or make predictions. Networks may require modeling billions of nodes and their relationships [3]. Nevertheless, when graphs get very large, working with them becomes challenging. Among techniques that reduce their size while preserving properties of relevance, we find graph sampling and graph summarization [14, 24]. While graph sampling considers a subset of nodes and edges from the original graph, graph summarization reduces the graph to a smaller data structure that maintains the relevant properties. With the increasing relevance of graph neural networks, graph sampling has regained new relevance: sampling methods have become an indispensable strategy to speed up their training [6, 23, 34].

When applying graph sampling techniques, attention must be devoted to whether particular sampling techniques preserve relevant aspects and information of the graph relevant to downstream tasks. While some research was devoted to understanding how graph sampling techniques affect a specific graph, the variety of tasks and aspects to be considered make it an open and relevant research question [20, 23, 29, 39].

*Motivation.* Multiple libraries have been developed for network analysis and operation. NetworkX [13] has long been a reference library for graph processing, implementing a wide range of generators and graph processing algorithms. In the same line, the Stanford Network Analysis Project (SNAP) library [21] has been developed to provide efficient implementations for graph processing at scale and collecting relevant network datasets. Among distributed processing implementations, we find the GraphX module, which was implemented on the top of Apache Spark [11]. Nevertheless, in contrast to the abovementioned libraries, it provides a narrow selection of implementations of graph processing algorithms. More recently, standards have been developed for graph frameworks to ensure standard building blocks for expressing graph algorithms in the language of linear algebra (e.g., GraphBLAS [4]), and several implementations were developed for them (e.g., SuiteSparse [7] or GraphBLAST [37]). In addition, multiple libraries have been developed for deep learning on graphs (e.g., Deep Graph Library [36] and PyTorch Geometric [8]. Nevertheless, when it comes to graph sampling, while particular sampling algorithms have been released (e.g., GraphSAINT [40]), few libraries gather graph sampling algorithms in a single package. One such library is *Little Ball of Fur* [32], which provides a Python interface and implementations for over twenty sampling techniques. We propose Go-Network, a library for efficient network generation and sampling, to address this void in the Go language.

*Contribution.* We present Go-Network, an open-source graph generation and sampling library implemented in Go. We describe the library design and implementation, highlighting particular goals and choices.

*Outline.* The paper has four sections. Section 2 briefly describes our choice for the Go language. Section 3 reviews graph sampling methods. Section 4 describes the library implementation, design

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652903

choices, and the implemented graph sampling algorithms. Finally, Section 5 concludes the paper and outlines future research.

## 2 GO LANGUAGE FOR MACHINE LEARNING DEVELOPMENT

The Go programming language was designed and introduced by Google in 2009 as a statically typed and compiled programming language, prioritizing simplicity, safety, and concurrency. While faster than, e.g., Python, it is currently an overlooked language for the development of machine learning libraries, mainly due to its lack of native support for CUDA and the limited amount of specialized libraries focused on statistics, calculus, and matrix manipulation - key to efficient machine learning implementations. Nevertheless, its simplicity and ease of implementing concurrency make it an attractive option for developing multi-threaded implementations requiring overly complex code, e.g., if developed in C++. We expect that, with time, the current shortcomings of the Go language ecosystem will be overcome, making it the go-to option for machine learning library development.

## 3 RELATED WORK

This section briefly describes graph sampling algorithms, focusing on the subset considered in two graph sampling libraries: *Little Ball of Fur* [32] and Go-Network.

### 3.1 Graph sampling

We consider graph sampling methods to be divided into categories based on two aspects: (a) the node/link selection criteria and (b) the operation applied to the graph upon node/link selection. The node/link selection criteria are usually divided into node-, link-, exploration-based, and hybrid methods. On the other hand, three operations can be applied to the graph: node/link preservation, contraction, or deletion. Following this taxonomy, we briefly summarize thirty graph sampling methods (see Table 1), and present them in detail in the following subsections.

Much effort has been invested into characterizing the graph sampling methods to understand what properties from the source graph are preserved in the graph sample [19, 20, 35, 38]. In particular, Krishnamurthy et al. [16] have shown that deletion or contraction methods allow resampling graphs to about 70% of their original size while keeping some original graph properties (e.g., the power-law distribution is respected). The author highlighted that among the benefits of resampling are simulation speed-ups: the authors estimated that reducing a graph to up to 70% of the original size can lead to simulation speed-ups of 11x or 37x for $O(n^2)$ or $O(n^3)$ simulations. For a detailed overview of sampling techniques and their properties, we defer the reader to the surveys by Hu et al. [14] Qi [28], and Liu et al. [23].

*3.1.1 Node/link contraction.* Sampling by node/link contraction involves iteratively merging edges in a large graph, reducing its size while preserving key properties. This process continues until a representative sample is obtained, enabling efficient analysis and exploration of the original graph's structure and characteristics. We depict the procedure in Fig. 1.

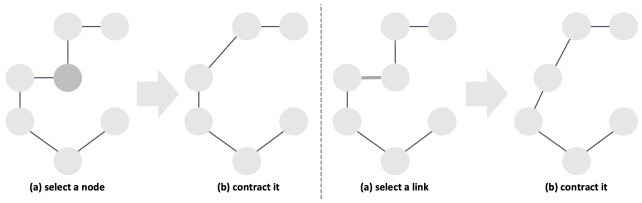



**Figure 1: Graph sampling by contraction works by incrementally contracting (merging) nodes/links from a graph. The Figure depicts two types of contractions node-based (on the left) and link-based (on the right). The selected node/link is colored in dark gray.**

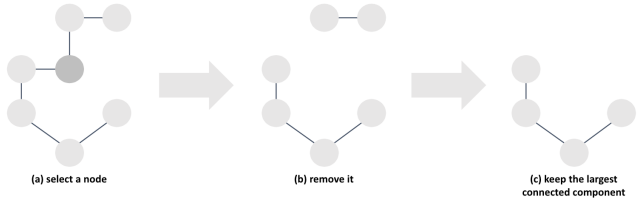



**Figure 2: Graph sampling by deletion works by incrementally removing nodes/links from a graph. The Figure depicts a case of node-based deletion sampling. The selected node is colored in dark gray.**

*3.1.2 Node/link deletion.* Sampling by node/link deletion is meant to gradually remove nodes/links from the graph until the desired graph size is achieved. This technique was introduced by Krishnamurthy et al. [16], who considered such sampling should follow three steps: (i) select nodes/links to be removed (only a small percentage (3%-5%) of nodes/links) and delete them, (ii) compute the connected components, preserving the largest and deleting the rest, and (iii) restart the procedure until achieving the desired graph size. We consider (ii) to be done to ensure the resulting graph mirrors a property observed in real-world graphs: that all of their elements are linked [3]. We depict the procedure in Fig. 2.

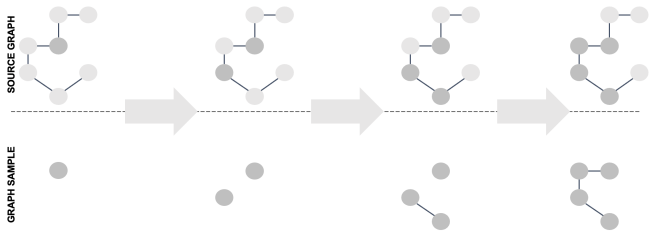



**Figure 3: The Figure depicts a case of node-based preservation sampling. The selected nodes are colored in dark gray.**

*3.1.3 Node/link preservation.* While the deletion methods select nodes and edges to delete them from the graph, preservation methods do the opposite: they retain them. Graph sampling by preservation works by incrementally adding nodes/links from the source graph to a new (initially empty) one. While the deletion-based methods follow a procedure that ensures the resulting graph has a single connected component, the preservation methods cannot provide such guarantees. We depict the procedure in Fig. 3.

| Node/link | Selection criteria | Operation applied to node/link selection | | |
|---|---|---|---|---|
| | | Contraction | Deletion | Preservation |
| Node | Breadth First Search | | | |
| | Circulated Neighbors Random Walk | | | [41] |
| | Common Neighbor Aware Random Walk | | | [22] |
| | Community Structure Expansion | | | [25] |
| | Depth First Search | | | |
| | Diffusion | | | [33] |
| | Diffussion Tree | | | [33] |
| | Forest Fire | | | [20] |
| | Frontier | | | [30] |
| | Inclusive Random Neighbour | | | [26] |
| | Loop Erased Random Walk | | | [17] |
| | Metropolis Hastings Random Walk | | | [15] |
| | Non-Back Tracking Random Walk | | | [18] |
| | PageRank | | | [20] |
| | Random | | [16] | [35] |
| | Random Degree | | [16] | [1] |
| | Random Neighbour | | | [5] |
| | Random Walk | | | [20] |
| | Random Walk With Jump | | | [20] |
| | Random Walk With Restart | | | [20] |
| | Shortest Path | | | |
| | Snowball | | | [12] |
| | SpikyBall | | | [31] |
| Link | Hybrid of RL and RNL | | [16] | |
| | Inclusive Random Node/Link | | | [26] |
| | Random Link | [16] | [16] | |
| | Random Link With Induction | | | [2] |
| | Random Link With Partial Induction | | | [2] |
| | Random Node/Link | [16] | [16] | |
| | Random Walk | | | [20] |

**Table 1: The table lists various graph sampling methods, referencing the scientific works in which they were introduced.**

*3.1.4 Node/link selection strategies.* Multiple strategies have been devised to perform node/link selection. In this section, we introduce some of them. In Table 1, we reference the scientific works in which they were introduced, considering their intersection with the operations applied upon node/link selection. Below, we briefly introduce each of them:

- **Breadth-First Search**: starting at a random node, it performs breadth-first search, including all of the nodes/links traversed until achieving the desired size;
- **Circulated Neighbors Random Walk**: simulates a random walker where the nodes of a neighborhood are randomly shuffled to ensure the walker can escape from closely knit communities;
- **Common Neighbor Aware Random Walk**: simulates a random walker that has a preference for neighbors with a lower number of common neighbors;
- **Community Structure Expansion**: given a random node from the graph, it chooses a node already connected to existing sampled nodes, always generating a connected graph;
- **Depth First Search**: starting at a random node, it performs depth-first search, including all of the nodes/links traversed until achieving the desired size;
- **Diffusion**: simulates a diffusion process, sampling nodes/links affected by the process;
- **Diffussion Tree**: is initiated by selecting a random node and expanding via random walks to neighboring nodes. It aims to efficiently capture local neighborhood structures while preserving connectivity, yielding a representative subgraph for analysis or further sampling;
- **Forest Fire**: simulates the fire spread through a forest, where each node represents a tree and edges represent potential paths of fire propagation. Starting from a randomly chosen node, it iteratively spreads fire to neighboring nodes based on a predefined probability parameter, typically resulting in a graph structure characterized by clusters and long-range connections resembling a forest fire propagation pattern;
- **Frontier**: iteratively expands a frontier set of nodes that consists of nodes adjacent to the current subgraph. This method efficiently explores the graph structure, allowing for the generation of representative subgraphs for various graph analysis tasks such as clustering, community detection, or pattern mining;
- **Inclusive Random Neighbour**: similar to random neighbor sampling, it includes the random node and the sampled neighbor into the sampled graph;
- **Loop Erased Random Walk**: is a stochastic algorithm used to generate random spanning trees on graphs. It operates by performing a random walk on the graph, erasing loops encountered during the walk, and ultimately constructing a spanning tree of the graph based on the path traversed without forming cycles;
- **Metropolis Hastings Random Walk**: is a Markov Chain Monte Carlo method for sampling graphs based on a target distribution. It iteratively explores the space of possible graphs by proposing changes to the current graph state and accepting or rejecting these changes based on a probability criterion derived from the Metropolis-Hastings algorithm, ensuring convergence to the desired distribution;
- **Non-Back Tracking Random Walk**: generates random walks on a graph where the walker does not backtrack to its previous node at the next step, ensuring a path without repetition. This technique is often employed in graph analysis and sampling to explore the structure of the graph efficiently while avoiding redundant traversal;
- **Non-Back Tracking Random Walk**: samples selecting nodes with a probability proportional to their PageRank scores, preserving the structural properties essential for PageRank calculations;
- **Random**: randomly selects a node/link from the graph;
- **Random Degree**: randomly selects a node from the graph with a probability proportional to their degree, ensuring that the resulting graph maintains similar connectivity patterns to the original one;
- **Random Node**: randomly selects a node from the graph and then switches it for a randomly sampled neighbor;
- **Random Walk**: simulates a random walk through its nodes and edges. It involves starting from a random node, then iteratively moving to neighboring nodes according to a stochastic process, resulting in a sequence of visited nodes that represent a sample from the graph's structure;
- **Random Walk With Jump**: similar to the Random Walk, but for each step, a decision is made with a probability of c=0.15 whether to continue the random walk or to jump to some random node within the graph;
- **Random Walk With Restart**: similar to the Random Walk, but for each step, a decision is made with a probability of c=0.15 whether to continue the random walk or to the starting node;
- **Random Walk With Restart**: starts by selecting two random nodes to compute the shortest path between them later;

- **Snowball**: iteratively samples nodes based on their connectivity to previously sampled nodes, expanding the sampling radius in a snowball-like manner. It starts with a small set of seed nodes and gradually adds nodes connected to the sampled nodes, typically used for capturing local neighborhoods in large graphs efficiently;
- **SpikyBall**: constructs graphs by placing nodes on the surface of a high-dimensional sphere and connecting them based on geometric rules, resulting in graphs with a spiky appearance.

# 4 LIBRARY DESIGN AND IMPLEMENTATION

## 4.1 Implemented methods

In Table 2, we provide a matrix describing the implemented graph sampling methods, grouping them based on whether the sampled element is a node or link, and two dimensions: (a) graph node/link selection criteria and (b) the operation applied on the graph upon the selected node/link.

| Node/link | Selection criteria | Operation applied to node/link selection | | |
|---|---|---|---|---|
| | | Contraction | Deletion | Preservation |
| Node | Breadth First Search | | | BoF |
| | Circulated Neighbors Random Walk | | | BoF |
| | Common Neighbor Aware Random Walk | | | BoF |
| | Community Structure Expansion | | | BoF |
| | Depth First Search | | | BoF |
| | Diffusion | | | BoF |
| | Difussion Tree | | | BoF |
| | Forest Fire | | | BoF |
| | Frontier | | | BoF |
| | Inclusive Random Neighbour | GoN | GoN | GoN |
| | Loop Erased Random Walk | | | BoF |
| | Metropolis Hastings Random Walk | | | BoF |
| | Non-Back Tracking Random Walk | | | BoF |
| | PageRank | | | BoF |
| | Random | GoN | GoN | BoF, GoN |
| | Random Degree | GoN | GoN | BoF, GoN |
| | Random Neighbour | GoN | GoN | BoF, GoN |
| | Random Walk | GoN | GoN | BoF, GoN |
| | Random Walk With Jump | GoN | GoN | BoF, GoN |
| | Random Walk With Restart | GoN | GoN | BoF, GoN |
| | Shortest Path | | | BoF |
| | Snowball | | | BoF |
| | SpikyBall | | | BoF |
| Link | Hybrid of RL and RNL | GoN | GoN | BoF, GoN |
| | Inclusive Random Node/Link | | | |
| | Random Link | GoN | GoN | BoF, GoN |
| | Random Link With Induction | | | BoF |
| | Random Link With Partial Induction | | | BoF |
| | Random Node/Link | GoN | GoN | BoF, GoN |
| | Random Walk | | | |

**Table 2: The table lists a variety of graph sampling methods, indicating which ones are supported by Go-Network (GoN) and which ones by the *Little Ball of Fur* library.**

## 4.2 Main modules

*Overview.* Go-Network is written in the Go language. The library declares a core graph model package and defines a graph interface. The current release only supports undirected graphs. The graph nodes and edges are modeled as integer values. Three kinds of utilities have been made available so far: (a) data loading and persistence, (b) graph generation, and (c) graph sampling.

*Data loading and persistence.* Various utilities are provided to load and persist graphs encoded in various formats. We find edge lists and adjacency lists among the supported formats.

*Graph sampling.* Thirty graph sampling algorithms were implemented. To ensure extensibility regarding graph sampling algorithm implementations, a Visitor pattern [10] was used. Doing so allows new methods to be seamlessly included without changing existing graph interfaces.

## 4.3 Design choices

To support a wide range of graph sampling algorithms while keeping a stable graph interface, we implemented a Visitor pattern [27] invoked on a graph. In particular, the sampling strategy is implemented as a Visitor and calls upon a graph, which provides itself to the Visitor to perform the sampling and then return the sampled graph. Given certain sampling algorithms follow the same structure (e.g., deletion sampling strategies perform incremental deletions and select the biggest connected component), such structure was implemented as a Template pattern [9], ensuring only the relevant sampling strategy is provided, reusing the rest of the code from the base struct. In Go-Network, graph sampling is not meant to be destructive. Therefore, a deep copy of the original graph is created before a sampling operation starts with deletion or contraction sampling strategies. Each time sampling is invoked on a graph, a new graph instance will be returned. This choice was made to ensure (i) immutability, given immutable data is implicitly concurrent-safe, and each concurrent process may operate on the same data without modifying it, and (ii) once a graph is loaded, multiple graph sampling algorithms can be applied simultaneously without altering it while obtaining the corresponding sampled graphs. This design decision may be reviewed in the future based on our benchmarking experience performed on massive graphs.

# 5 CONCLUSION AND FUTURE WORK

In this paper, we have introduced Go-Network and compared it against another graph sampling library: *Little Ball of Fur*. Go-Network does not cover the wide range of node/link selection strategies *Little Ball of Fur* yet provides. Nevertheless, Go-Network can already boast more sampling strategies, providing implementations that span graph sampling by contraction, deletion, and preservation, while *Little Ball of Fur* provides only implementations considering graph preservation. We consider supporting graph contraction and deletion as key to scalable graph sampling while ensuring the graph nodes remain connected [16]. Future work will focus on three areas: (a) enrich data loading and graph generation capabilities, (b) implement additional graph sampling techniques while adding support for multi-threaded and distributed execution, and (c) benchmark the graph sampling algorithms while comparing them against other implementations.

# ACKNOWLEDGMENTS

Graph-Massivizer receives funding from the Horizon Europe research and innovation program of the European Union. Its grant management number is 101093202:https://graph-massivizer.eu/.

# REFERENCES

[1] Lada A Adamic, Rajan M Lukose, Amit R Puniyani, and Bernardo A Huberman. 2001. Search in power-law networks. *Physical review E* 64, 4 (2001), 046135.

[2] Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. 2013. Network sampling: From static to streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 8, 2 (2013), 1–56.

[3] Albert-László Barabási and Márton Pósfai. 2016. *Network science*. Cambridge University Press, Cambridge. http://barabasi.com/networksciencebook/

[4] Benjamin Brock, Aydın Buluç, Timothy Mattson, Scott McMillan, and José Moreira. 2019. The graphblas c api specification. *GraphBLAS. org, Tech. Rep* (2019).

[5] Reuven Cohen, Shlomo Havlin, and Daniel Ben-Avraham. 2003. Efficient immunization strategies for computer networks and populations. *Physical review letters* 91, 24 (2003), 247901.

[6] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. 2020. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1393–1403.

[7] Timothy A Davis. 2019. Algorithm 1000: SuiteSparse: GraphBLAS: Graph algorithms in the language of sparse linear algebra. *ACM Transactions on Mathematical Software (TOMS)* 45, 4 (2019), 1–25.

[8] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1993. Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP'93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings 7*. Springer, 406–431.

[10] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.

[11] Joseph E Gonzalez, Reynold S Xin, Ankur Dave, Daniel Crankshaw, Michael J Franklin, and Ion Stoica. 2014. {GraphX}: Graph processing in a distributed dataflow framework. In *11th USENIX symposium on operating systems design and implementation (OSDI 14)*. 599–613.

[12] Leo A Goodman. 1961. Snowball sampling. *The annals of mathematical statistics* (1961), 148–170.

[13] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[14] Pili Hu and Wing Cheong Lau. 2013. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865* (2013).

[15] Christian Hübler, Hans-Peter Kriegel, Karsten Borgwardt, and Zoubin Ghahramani. 2008. Metropolis algorithms for representative subgraph sampling. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 283–292.

[16] Vaishnavi Krishnamurthy, Michalis Faloutsos, Marek Chrobak, Li Lao, J-H Cui, and Allon G Percus. 2005. Reducing large internet topologies for faster simulations. In *International Conference on Research in Networking*. Springer, 328–341.

[17] Gregory F Lawler. 1999. Loop-erased random walk. *Perplexing Problems in Probability: Festschrift in Honor of Harry Kesten* (1999), 197–217.

[18] Chul-Ho Lee, Xin Xu, and Do Young Eun. 2012. Beyond random walk and metropolis-hastings samplers: why you should not backtrack for unbiased graph sampling. *ACM SIGMETRICS Performance evaluation review* 40, 1 (2012), 319–330.

[19] Sang Hoon Lee et al. 2006. Statistical properties of sampled networks. *Physical review E* 73, 1 (2006), 016102.

[20] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 631–636.

[21] Jure Leskovec and Rok Sosič. 2016. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1–20.

[22] Yongkun Li, Zhiyong Wu, Shuai Lin, Hong Xie, Min Lv, Yinlong Xu, and John CS Lui. 2019. Walking with perception: Efficient random walk sampling via common neighbor awareness. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 962–973.

[23] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. 2021. Sampling methods for efficient training of graph convolutional networks: A survey. *IEEE/CAA Journal of Automatica Sinica* 9, 2 (2021), 205–234.

[24] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph summarization methods and applications: A survey. *ACM computing surveys (CSUR)* 51, 3 (2018), 1–34.

[25] Arun S Maiya and Tanya Y Berger-Wolf. 2010. Sampling community structure. In *Proceedings of the 19th international conference on World wide web*. 701–710.

[26] Yitzchak Novick and Amotz Bar-Noy. 2023. Inclusive random sampling in graphs and networks. *Applied Network Science* 8, 1 (2023), 56.

[27] Jens Palsberg and C Barry Jay. 1998. The essence of the visitor pattern. In *Proceedings. The Twenty-Second Annual International Computer Software and Applications Conference (Compsac'98)(Cat. No. 98CB 36241)*. IEEE, 9–15.

[28] Xiao Qi. 2022. A Review: Random Walk in Graph Sampling. *arXiv preprint arXiv:2209.13103* (2022).

[29] Amir H Rasti, Mojtaba Torkjazi, Reza Rejaie, D Stutzbach, N Duffield, and W Willinger. 2008. Evaluating sampling techniques for large dynamic graphs. *Univ. Oregon, Tech. Rep. CIS-TR-08* 1 (2008).

[30] Bruno Ribeiro and Don Towsley. 2010. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 390–403.

[31] Benjamin Ricaud, Nicolas Aspert, and Volodymyr Miz. 2020. Spikyball sampling: Exploring large networks via an inhomogeneous filtered diffusion. *Algorithms* 13, 11 (2020), 275.

[32] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. Little ball of fur: a python library for graph sampling. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 3133–3140.

[33] Benedek Rozemberczki and Rik Sarkar. 2018. Fast sequence-based embedding with diffusion graphs. In *Complex Networks IX: Proceedings of the 9th Conference on Complex Networks CompleNet 2018 9*. Springer, 99–107.

[34] Marco Serafini and Hui Guan. 2021. Scalable graph neural network training: The case for sampling. *ACM SIGOPS Operating Systems Review* 55, 1 (2021), 68–76.

[35] Michael PH Stumpf, Carsten Wiuf, and Robert M May. 2005. Subnets of scale-free networks are not scale-free: sampling properties of networks. *Proceedings of the National Academy of Sciences* 102, 12 (2005), 4221–4224.

[36] Minjie Yu Wang. 2019. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*.

[37] Carl Yang, Aydın Buluç, and John D Owens. 2022. GraphBLAST: A high-performance linear algebra-based graph framework on the GPU. *ACM Transactions on Mathematical Software (TOMS)* 48, 1 (2022), 1–51.

[38] Sooyeon Yoon et al. 2007. Statistical properties of sampled networks by random walks. *Physical Review E* 75, 4 (2007), 046114.

[39] Muhammad Irfan Yousuf, Izza Anwer, and Raheel Anwar. 2023. Empirical characterization of graph sampling algorithms. *Social Network Analysis and Mining* 13, 1 (2023), 66.

[40] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

[41] Zhuojie Zhou, Nan Zhang, and Gautam Das. 2015. Leveraging history for faster sampling of online social networks. *arXiv preprint arXiv:1505.00079* (2015).

# Performance Optimization in the LLM World 2024

Kingsum Chow
College of Software Technology
Zhejiang University
Ningbo, Zhejiang, China
kingsum.chow@gmail.com

Yu Tang
College of Software Technology
Zhejiang University
Ningbo, Zhejiang, China
y.tang@zju.edu.cn

Zhiheng Lyu
Department of Computer Science
University of Hong Kong
Hong Kong SAR, China
cogito@connect.hku.hk

Anil Rajput
Datacenter Ecosystem
AMD Corporation
Portland, Oregon, USA
Anil_Rajput@yahoo.com

Khun Ban
Datacenter and AI
Intel Corporation
Hillsboro, Oregon, USA
khunban@gmail.com

## ABSTRACT

The popularity and adoption of large language models (LLM) like ChatGPT has evolved rapidly. LLM pre-training is expensive. ChatGPT is estimated to cost over $700,000 per day to operate and using GPT-4 to support customer service can cost a small business over $21,000 a month. The high infrastructure and financial costs, coupled with the specialized talent required, make LLM technology inaccessible to most organizations. For instance, the up-front costs include the emissions generated to manufacture the relevant hardware and the cost to run that hardware during the training procedure, both while the machines are operating at full capacity and while they are not. The best estimate of the dynamic computing cost in the case of GPT-3, the model behind the original ChatGPT, is approximately 1,287,000 kWh, or 552 tons of carbon dioxide. The goal of this workshop is to address the urgency of reducing energy consumption of LLM applications, by bringing together researchers from the academia and industry to share their experience and insights in performance engineering in the LLM world.

**ACM Reference format:**

Kingsum Chow, Yu Tang, Zhiheng Lyu, Anil Rajput and Khun Ban. 2024. Performance Optimization in the LLM World. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE'24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3629527.3651436

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright is held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651436

## Organiers/presenter and affiliations (including short bios)

**Kingsum Chow** (kingsum.chow@gmail.com) is a professor at the School of Software Technology, Zhejiang University. He received his Ph.D. in Computer Science and Engineering at the University of Washington in 1996. Prior to joining Zhejiang University in 2023, Kingsum has been working as a chief scientist and senior principal engineer in the industry. He has extensive experience in software hardware co-optimization from thirty years of working at Intel and Alibaba. He delivered two QCon keynotes. He appeared four times in JavaOne keynotes. He has been issued 30 patents. He has delivered more than 100 technical presentations. He has collaborated with many industry groups, including groups at Alibaba, Amazon, AMD, Ampere, Appeal, Arm, BEA, ByteDance, Facebook, Google, IBM, Intel, Microsoft, Netflix, Oracle, Siebel, Sun, Tencent and Twitter. In his spare time, he volunteers to coach multiple robotics teams to bring the joy of learning Science, Technology, Engineering and Mathematics to the K-12 students in USA and China.

**Yu Tang** (y.tang@zju.edu.cn) is a postgraduate student at the Zhejiang University. His advisor is Kingsum Chow. His research interest focuses on Large Language Model, system performance analysis and optimization.

**Zhiheng Lyu** (cogito@connect.hku.hk) is a distinguished senior at the University of Hong Kong (HKU) and concurrently serves as a research assistant within the Berkeley AI Research Lab at the University of California, Berkeley. His academic journey is punctuated by seminal contributions at both UCB and ETH Zürich, resulting in key papers on LLM interpretability and practical applications — several of which are under active conference review. A noteworthy internship at Megvii's R-face Institute allowed him to advance automatic CV model training systems, setting new industry standards. Beyond research, Zhiheng's prowess in algorithmic competitions is evident: he boasts two gold medals from regional ICPC events and an impressive appearance in the ICPC

world finals. As the current leader of the HKU AI4Good Community, Zhiheng consistently synthesizes his deep knowledge in AI, robotics, and LLMs to spur innovation. His role as the organizer of this workshop underscores his dedication to fostering richer insights into LLM application and optimization.

**Anil Rajput** (Anil_Rajput@yahoo.com) is an AMD Fellow, Software System Design, as core architect for datacenter and cloud with focus on performance, deployments, optimizations, and best practices. He received his certification in data analytics from Harvard Business Analytics Program in 2022 and his Master's in Electrical and Computer Engineering from Portland State University in 1997. Currently, Anil's focus areas are workloads characterization, platform evaluation, cloud deployments, on-prem datacenters as well as understanding and resolving large deployment issues at scale for critical customers. Earlier, he has been at Intel Corporation for more than 20 years, playing various roles in the Software and Services Group, leading platform design, managed runtime like Java and .Net, scripting languages and development of representative benchmarks as chair of Java committee at SPEC. He was key members of teams who architected and developed several benchmarks like SPECjbb2005, SPECjvm2008, SPECjEnterprise2010, SPECpower_ssj2008 etc. Anil is also guiding graduate students as mentor and also participates in local High School science fairs to encourage kids for STEM in Oregon, USA.

**Khun Ban** (khunban@gmail.com) is an Intel cloud performance architect leading a team to drive solutions to solve today's complex business problems by analyzing the requirements and making architecture recommendations for CPUs/storage/network balance to best meet the needs based on the constraints. He has over twenty years of enterprise software development experience. His current focus is on Open-Source Relational Databases. He received his B.S. degree in Computer Science and Engineering from the University of Washington in 1995.

## A list of topics to be covered, including format (e.g., talks, demos, etc.), target audience, and pre-requisite knowledge

The half day workshop will be composed of invited talks, work in progress and fully refereed papers and a panel. Presentations are not limited to the following topics:

1. Optimizing LLM Workloads on Traditional and New Architectures
   - Hardware Assisted LLM Systems
   - LLM Optimization at Scale
   - Code generation optimization for modern hardware
2. Panel Discussion (speakers from the industry and academia)

The target audience:

1. Researchers that are advocating new ways of optimizing LLM applications in software or hardware optimizations.
2. Practitioners that need to solve runtime performance problems in their LLM deployments.

## Expected duration : half day

## Expected attendees: 30

## The main organizer delivered the following workshops and tutorials in the past

- Runtimes in the Cloud 3, a full day workshop at HPCA, 2020/02, 20 attendees.
- Runtimes in the Cloud 2, a full day workshop at ISCA, 2019/06, 20 attendees.
- Runtimes in the Cloud, a full day workshop at ISCA, 2018/06, 30 attendees.
- Scaling Software Performance and Software Performance in the Cloud, a half day workshop at PNSQC, 2017/10, 50 attendees.
- Software Performance Analytics in the Cloud, a full day tutorial at ICPE, 2017/04, 20 attendees.
- Applying Analytics to Data Center Performance, a half day workshop at CMG Performance and Capacity Conference, 2015/11, 30 attendees.

## Workshop Website

https://sites.google.com/view/pollmw

# EchoSwift

## An Inference Benchmarking and Configuration Discovery Tool for Large Language Models (LLMs)

Karthik Krishna
CTO
InfobellIT Solutions Pvt. Ltd
Bengaluru, Karnataka, India
karthik@infobellit.com

Ramana Bandili
CEO
InfobellIT Solutions Pvt. Ltd
Bengaluru, Karnataka, India
braman@infobellit.com

## ABSTRACT

Large Language Models (LLMs) are advanced natural language processing models that are trained on vast amounts of text data to understand and generate human-like language. These models are designed to understand context, generate coherent and contextually relevant text, and demonstrate advanced language capabilities. In the dynamic landscape of LLMs, the demand for efficient inference benchmarking is crucial.

Organizations such as TPC and SPEC brought several industry standard benchmarks [1][2][3][4]. This publication introduces EchoSwift [11], a comprehensive benchmarking framework designed to evaluate the real-time performance of LLMs in deployment scenarios.

As LLMs ascend to the forefront of technological innovation, their seamless integration into real-world applications demands a nuanced understanding of their efficiency, throughput, latency, and scalability. It is within this dynamic landscape that our publication unveils the EchoSwift, a novel benchmarking framework meticulously crafted to address the pressing need for comprehensive inference benchmarking, as well as the discovery of the right configuration for specific LLM requirements. For instance, certain deployments might have 32 tokens as input and 256 tokens as output, while others might have 256 tokens as input and 64 tokens as output. It is crucial to acknowledge that the configuration for these two requirements need not be the same for an optimal performance, scale and better TCO. The EchoSwift not only aids in comprehensive configuration discovery but also facilitates robust Performance/Scale testing, ensuring that LLM deployments are not only efficient but also finely tuned to their specific operational demands.

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
*ICPE Companion '24 , May 7–11, 2024, London, United Kingdom*
© 2024 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652273

## CSS CONCEPTS

• **Computer Systems Organization → Artificial Intelligence → Natural Language Processing.**

## KEYWORDS

Large language models, Text generation Inference, Llama2, LLM Performance, AI Benchmarking

**ACM Reference format:**

Karthik Krishna and Ramana Bandili. 2024. EchoSwift: An Inference Benchmarking and Configuration Discovery Tool for Large Language Models (LLMs), 2024. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering. May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA. https://doi.org/10.1145/3629527.3652273

## 1. INTRODUCTION

LLMs have become so profound that language comprehension and production have transcended traditional boundaries, making it imperative to gauge the real-time performance of these models in deployment scenarios more crucial than ever. The advent of LLMs, exemplified by models like the Llama2 from Meta with varying parameters and precision levels, has propelled them into the core of applications ranging from natural language processing to AI-driven services.

This publication delves into the intricate challenges posed by diverse LLM variants. Llama2 is one such open sourced publicly available LLM and this benchmarking tool was primarily tested with Llama2, however, this tool is applicable to all different LLMs deployed with various architectures and technologies. Llama2 is an advanced AI platform that combines cutting-edge algorithms, extensive data sets, and powerful computational capabilities to deliver exceptional results. Llama2 model has various models which different in parameters such as 7B, 13B, and 70B, coupled with precision nuances in BF16, Int8, and Int4. These intricacies make the identification of an ideal and efficient infrastructure for serving these models a formidable challenge. Enter EchoSwift – a compass guiding practitioners through the delicate balance between model complexity and operational efficiency in the realm of LLMs.

In this publication, we embark on a journey to introduce and expound upon EchoSwift, a benchmarking framework tailored to

assess the real-time performance of LLMs. As we traverse through the subsequent sections, we unravel the significance of this framework, its methodology, and the pivotal role it plays in shaping the deployment landscape for Large Language Models.

## 2. BACKGROUND

Before the advent of LLMs, a substantial 70% of the AI Inference market was dominated by CPU architectures, highlighting the transformative shift brought about by the introduction of LLMs in the landscape of inference processing. Within the burgeoning landscape of LLMs, this publication unveils EchoSwift – a pioneering benchmarking framework meticulously crafted to assess the real-time performance of LLMs in deployment scenarios. The results presented here reflect out-of-the-box performance with currently released software, with the anticipation of additional performance gains in upcoming releases.

## 3. ECHOSWIFT OVERVIEW APPROACH

The article outlines benchmarking the performance of LLM using LLama2-7B as the sample LLM model and measures Token Latency, Throughput calculated as tokens per second, and Time To First Token (TTFT).

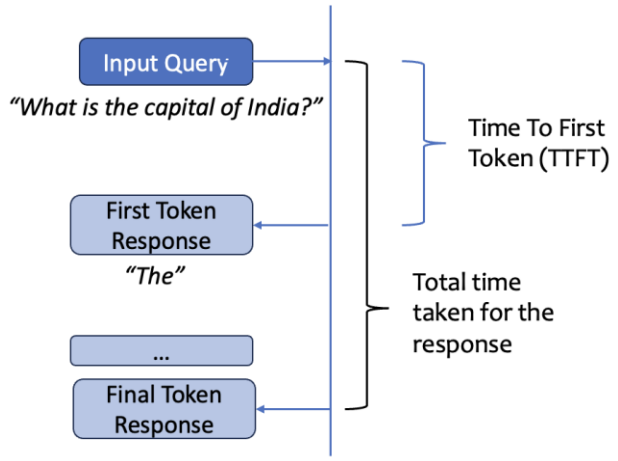



**Figure 1: Performance Metrics**

Latency is measured of time to output each token when streaming the output excluding the first token and is often measured in millisecond.

$$Latency = \frac{Total\ time\ to\ output - TTFT}{Total\ Number\ of\ Tokens - 1}$$

TTFT is the time to process the prompt and output the first token and is often measured in millisecond.

$$TTFT = Time\ To\ First\ Token$$

Throughput is calculated as tokens per second which takes in to account the total time taken to output all the tokens and normalized to 1 second, i.e., total tokens for the output divided by total time taken in seconds.

$$Throughput = \frac{Total\ No\ of\ tokens}{Total\ Time\ to\ Output\ (in\ seconds)}$$

The EchoSwift Benchmark is used in two modes:

a) Configuration Discovery Mode

b) Performance Benchmarking Mode

In Configuration Discovery mode, we restrict the number of parallel requests to 1, 3, or 10 while varying the parameters for input token size and output token size based on the specific requirements of the application. We employ this approach to test different scenarios and identify Token Latency, Throughput, and TTFT for various combinations of Input and Output tokens. The data obtained is then used to discover the optimal configuration.

In Performance Benchmarking Mode, we maintain the input token and output token size as constants (for example, 32 tokens for Input and 256 tokens for output or any other combination specific to the application requirement) and scale the number of parallel requests (or parallel users) for this fixed combination of a single input and output token. This scaling enables a better sizing of the environment.

## 4. BENCHMARKING METHODOLOGY AND STEPS

The steps for benchmarking LLM have been discussed below:

### 4.1. Data Collection

The Hugging Face Hub consists of the vast amount datasets for variety of domains and tasks. The datasets available on Hugging Face is continually expanding, and new datasets are consistently being added by both the Hugging Face team and the community. The Hugging Face Hub hosts a large number of community-curated datasets for a diverse range of domains, languages, and tasks such as translation, automatic speech recognition, and image classification.

Latency per token = (latency – TTFT) / (output tokens -1)

| request | start_time | end_time | input_tokens | output_tokens | latency(ms) | throughput(tokens/second) | time_per_token(ms/tokens) | TTFT(ms) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1970-01-08 17:25:27.448310 | 1970-01-08 17:25:37.612477 | 131 | 66 | 10164.167228969745 | 19.381814128216405 | 147.5228992004234 | 575.178780942224 |
| 2 | 1970-01-08 17:25:37.638074 | 1970-01-08 17:25:44.336981 | 130 | 40 | 6698.907856014557 | 25.37727099013117 | 151.86879279700895 | 776.0249369312078 |
| 3 | 1970-01-08 17:25:44.338007 | 1970-01-08 17:25:54.837021 | 127 | 66 | 10499.014172004536 | 18.382678300847676 | 149.71888535297833 | 767.2866240609437 |
| 4 | 1970-01-08 17:25:54.838166 | 1970-01-08 17:26:05.026627 | 125 | 66 | 10188.46081092488 | 18.746698205404545 | 147.41419543010684 | 606.5381079679355 |
| 5 | 1970-01-08 17:26:05.027506 | 1970-01-08 17:26:15.215437 | 131 | 66 | 10187.930912012234 | 19.336605410989208 | 146.22517707757652 | 683.2944019697607 |
| 1 | 1970-01-08 17:26:27.510252 | 1970-01-08 17:26:43.019906 | 132 | 94 | 15509.653392946348 | 14.571569994129126 | 160.0505927632693 | 624.9482659623027 |
| 2 | 1970-01-08 17:26:43.041832 | 1970-01-08 17:27:03.140747 | 126 | 130 | 20098.91493699979 | 12.737005992733142 | 150.68639124032515 | 660.370466997847 |
| 3 | 1970-01-08 17:27:03.141985 | 1970-01-08 17:27:22.713432 | 122 | 130 | 19571.4472719701 | 12.87590010580925 | 147.0961459531528 | 596.044444013387 |
| 4 | 1970-01-08 17:27:22.714595 | 1970-01-08 17:27:43.008450 | 127 | 130 | 20293.855173047632 | 12.663931904930658 | 152.5422360237269 | 615.9067259868607 |
| 5 | 1970-01-08 17:27:43.010358 | 1970-01-08 17:28:03.043577 | 127 | 129 | 20033.219030010514 | 12.778775074365353 | 151.77725606190506 | 605.7302540866658 |
| 1 | 1970-01-08 17:28:25.104480 | 1970-01-08 17:29:05.333811 | 127 | 258 | 40229.331478942186 | 9.570131688654236 | 154.1136843813692 | 622.114592930302 |
| 2 | 1970-01-08 17:29:05.355198 | 1970-01-08 17:29:45.507219 | 124 | 258 | 40152.020766050555 | 9.513842459530448 | 153.54457519837956 | 691.0649400670081 |
| 3 | 1970-01-08 17:29:45.508864 | 1970-01-08 17:30:25.934042 | 128 | 258 | 40425.17778498586 | 9.548504698063754 | 154.579172443584 | 698.3304669847712 |
| 4 | 1970-01-08 17:30:25.936447 | 1970-01-08 17:31:06.918849 | 122 | 258 | 40982.402284047566 | 9.272272458950395 | 157.11764341622504 | 603.167926077731 |
| 5 | 1970-01-08 17:31:06.920197 | 1970-01-08 17:31:43.705990 | 122 | 243 | 36785.79278790858 | 9.922308922480932 | 149.59577474337118 | 583.6153000127524 |

**Figure 2: Sample Output for varying combinations of input and output token for Single User**

The dataset used here in the benchmark is from ShareGPT Dataset from Hugging Face. Dataset has been filtered based on varying input token lengths. Considered input token lengths in this context range from 32 to 2,000, with variations of approximately ±10 tokens for each length. The specified lengths include 32, 64, 128, 256, 512, 1K and 2K tokens, providing a comprehensive coverage of input sizes for benchmarking. The dataset contains the 7 different files that have 1000 prompts for each token length as specified. Python file DatasetFiltering.py has been used here for Data Processing.

## 4.2. Configuration Discovery Test

The objective of the work involves identifying the optimal configuration with a single container test.

The analysis involves determining the optimal latency, throughput and TTFT by sending individual requests one at a time with

different input and output tokens. To enhance throughput, parallel requests are then dispatched to the endpoint. Figure 2 above depicts the sample output capturing the performance metrics when a single request for 128 input tokens and with varying output token combinations 64, 128, 256 is given as input request. Similarly varying combinations of input and output tokens in different combinations like 128 output tokens for 128 input and 256 output tokens for 128 inputs for 5 parallel requests are sent to capture the ideal performance parameters.

The maximum throughput is identified when the model consistently provides prompt responses to input requests without significant degradation in latency. This approach allows for a balanced assessment, ensuring that the system achieves optimal performance by striking the right balance between response time and concurrent processing capabilities.

## 4.3. Scale Testing/Parallel Requests

Locust Load testing has been used for benchmarking setup. Locust is an opensource load testing tool, written in Python and is a highly valuable tool for identifying performance bottlenecks, testing the scalability of system, and ensuring that the developed web applications can handle a specified level of traffic. The tool allows to set the Number of Users which indicates the maximum no. of users that can run simultaneously, and Spawn Rate denotes the number of users that will be spawned per second.

For deployment, hugging face text-generation inference model server 1.1.1 is used.

The steps below need to be followed to run the load test:

1. Define the configurations to run the load test.
2. Listing the parallel users (1, 3, 10, 30) and the Input tokens (32, 64, 128, 256, 512) and Output tokens (32, 64, 128, 256, 512).
3. TGI endpoint has been used for hosting the model.

In Section 5 the results are discussed and the generated graphs for performance metrics have been explained in detail.

# 5. RESULT ANALYSIS

The result analysis involves determining the optimal latency, throughput and TTFT by sending individual requests one at a time with different input and output tokens. To enhance throughput, parallel requests are then dispatched to the endpoint. This section gives the detailed observation for Configuration Discovery Result analysis and performance test.

## 5.1. Configuration Discovery Result Analysis

To identify an optimal Configuration to achieve ideal token latency and throughput, the systems are tested with various combinations of input and output tokens. The below graphs

illustrate the Throughput, Token latency and TTFT for single user sending the requests to the endpoint for 32, 64, 128 input tokens and 64,128, 256 output tokens.

In Figure 3, it can be observed that the throughput varies between 4.94 tokens/second to 11.88 tokens/second for single user.

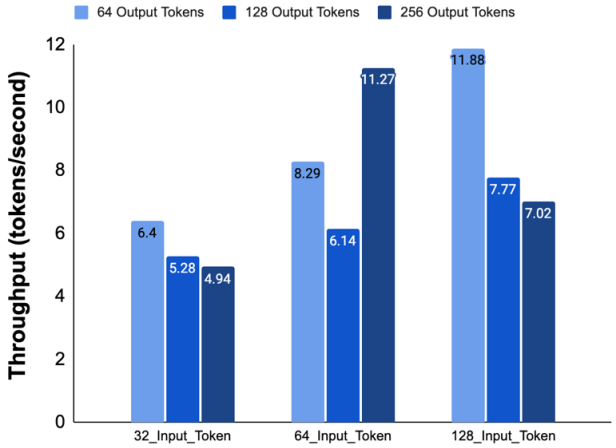



**Figure 3: Throughput for Single User**

Figure 4 depicts tokens latency for single user ranging from 233 milliseconds/token to 247 milliseconds/token.

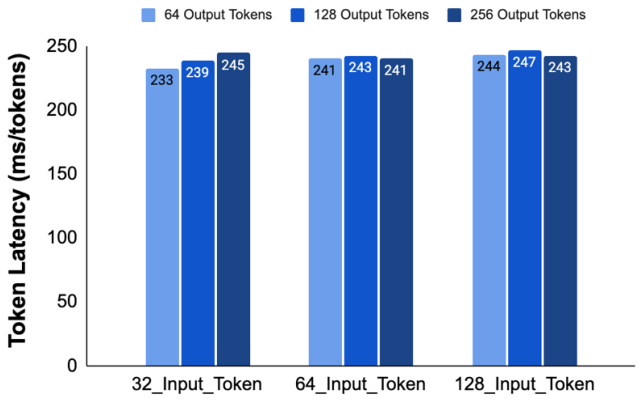



**Figure 4: Token Latency for Single User**

Similarly, Figure 5 depicts the TTFT for single user it varies between 363 milliseconds to 859 milliseconds.

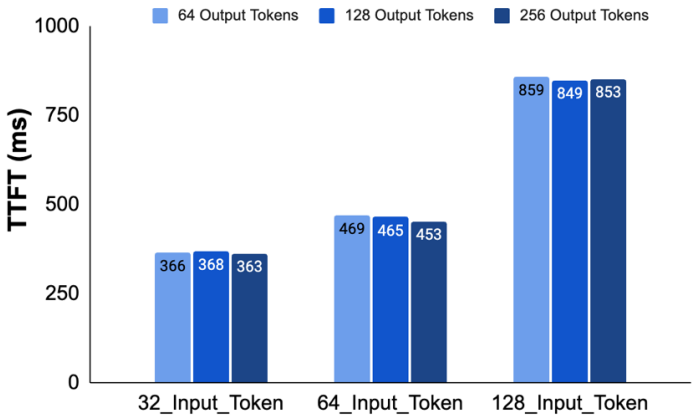



**Figure 5: TTFT for a Single User**

The achieved performance results were obtained through testing the model on a hardware configuration featuring a 16-core CPU and 128 GB of RAM. It is anticipated that conducting the same load testing on more robust hardware configurations will likely yield even more substantial improvements in performance.

## 5.2. Performance Test (with Parallel Requests) Result Analysis

The model can also be tested against multiple users for parallel requests sent to the model endpoint for varying input and output tokens combinations. Performance testing with parallel requests is a critical aspect of evaluating the robustness and scalability of a system. When analysing the results of such tests, it is important to consider various factors to gain insights into the system's behaviour under intense loads. Therefore, a comprehensive analysis of performance test is done by examining throughput and token latency against parallel requests sent to the model to get some insights for improvement.

Line graph shown in Figure 6 depicts the relationship between the number of parallel requests made to the model endpoint and the average latency of those requests. It can be observed that when number of parallel requests increases, the average latency also increases due to limited system resources. Thus, it is utmost important to identify the ideal configuration that can handle multiple parallel requests for scale testing.

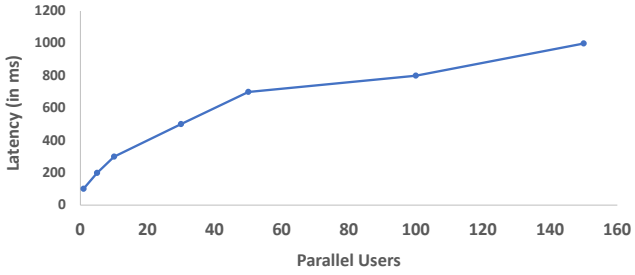



**Figure 6: Latency vs Parallel Requests**

Graph in Figure 7 shows the relationship between the number of parallel requests made to the model endpoint and the throughput of the system, measured in tokens per second. It can be inferred from the graph that the throughput increases linearly with the requests initially and starts slowing down as the resources become saturated, and eventually decreases when the system is overloaded as the system has limited resources.

Thus, the specific curve and values will vary depending on the specific system and workload, but the general trend is consistent.

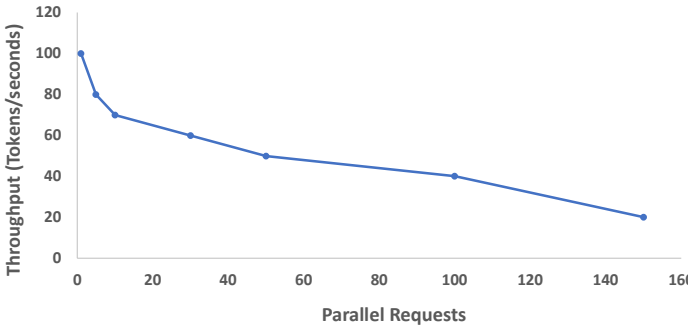



**Figure 7: Throughput vs Parallel Requests**

The above graphs can be used to understand the performance limitations of a system under increasing load. It can help the users to determine the optimal number of system configurations required to handle the concurrent requests to while maintaining acceptable throughput and latency. Additionally, it can be used to compare the performance of different systems or to track changes in performance over time.

## 6. CONCLUSION

This benchmark can be used to evaluate a single container, or a cluster with thousands or nodes deploying an LLM. This can be used to test scale, test latency, throughput and TTFT for any environment deploying an LLM. This is not limited to Llama2 but any form of LLM, quantized models with lower precisions (int8, int4, etc) and different precision and different sizes with and without CPU, GPU, Accelerators, or other technology.

This could also be used for inference benchmarking with Retrieval Augmented Generation (RAG) based applications, Fine Tuning models or Fully trained LLM models.

Benchmarking LLMs provides valuable insights for businesses aiming to deploy natural language processing applications. To make the best decisions, it's crucial to acknowledge the specific needs of each application and understand how well LLMs perform on different types of CPUs, GPUs and Accelerators to identify the ideal throughput, latency and scale and drive the total cost of ownership (TCO) lower. Consideration of the specific requirements of each application, coupled with an understanding of the strengths and weaknesses of LLMs on different software and hardware technologies, and architectures empowers businesses to make informed and optimised decisions.

In line with our commitment to standardization and industry best practices, we propose this workload to industry standard organizations like SPEC to create standards for Inference on Large Language Models. Establishing such standards will further facilitate benchmarking efforts, promote consistency, and provide a solid foundation for the broader adoption of LLMs in various applications.

## ACKNOWLEDGEMENTS

Authors would like to thank Anna Joseph, Gogula Akhil Reddy , Arun Kumar Tiwary , Bhavana k, Divya Singh, Harshitha T, Vadla Sai Charitha, Sarthak Dwivedi, Kammara Prasad Achari, Arunima Divya, who are engineers from InfobellIT who helped test and develop this benchmark.

## REFERENCES

[1] https://spec.org/
[2] https://tpc.org/
[3] Raghunath Nambiar, Tilmann Rabl, Karthik Kulkarni, Michael Frank:
Enhancing Data Generation in TPCx-HS with a Non-uniform Random Distribution. TPCTC: 2015: 94-129
[4] Meikel Poess, Raghunath Nambiar, Karthik Kulkarni, Chinmayi Narasimhadevara, Tilmann Rabl, Hans-Arno Jacobsen: Analysis of TPCx-IoT: The First Industry Standard Benchmark for IoT Gateway Systems. ICDE 2018: 1519-1530
[5] https://www.intel.com/content/www/us/en/developer/articles/technical/accelerate-llama2-ai-hardware-sw-optimizations.html
[6] https://huggingface.co/NousResearch/Llama-2-7b-hf?ref=blog.truefoundry.com
[7] https://github.com/huggingface/text-generation-inference
[8] https://locust.io/
[9] Llama 2: Open Foundation and Fine-Tuned Chat Models - https://arxiv.org/pdf/2307.09288.pdf
[10] https://www.anyscale.com/blog/reproducible-performance-metrics-for-llm-inference
[11] EchoSwift: https://github.com/Infobellit-Solutions-Pvt-Ltd/EchoSwift

# HotCloudPerf'24 Workshop Chairs' Welcome

Dragi Kimovski
dragi.kimovski@aau.at
University of Klagenfurt
Klagenfurt, Austria

Klervie Toczé
klervie.tocze@liu.se
Linköping University
Linköping, Sweden

Nikolas Herbst
nikolas.herbst@uni-wuerzburg.de
University of Würzburg
Würzburg, Germany

Tiziano De Matteis
t.de.matteis@vu.nl
Vrije Universiteit Amsterdam
Amsterdam, Netherlands

**ACM Reference Format:**
Dragi Kimovski, Klervie Toczé, Nikolas Herbst, and Tiziano De Matteis. 2024. HotCloudPerf'24 Workshop Chairs' Welcome. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3629527.3651415

It gives us immense pleasure to extend a warm welcome to you for the 2024 edition of the Workshop on Hot Topics in Cloud Computing Performance – HotCloudPerf 2024.

Cloud computing represents one of the most significant transformations in the realm of IT infrastructure and usage. The adoption of global services within public clouds is on the rise, and the immensely lucrative global cloud market already sustains over 1 million IT-related jobs. However, optimizing the performance and efficiency of the IT services provided by both public and private clouds remains a considerable challenge. Emerging architectures, techniques, and real-world systems entail interactions with the computing continuum, serverless operation, everything as a service, complex workflows, auto-scaling and -tiering, etc. The extent to which traditional performance engineering, software engineering, and system design and analysis tools can contribute to understanding and engineering these emerging technologies is uncertain. The community requires practical tools and robust methodologies to address the hot topics in cloud computing performance effectively.

In response to this demand, the HotCloudPerf workshop offers a platform for academics and practitioners in the field of cloud computing performance. The workshop seeks to foster engagement within this community and fosters the development of new methodological approaches to achieve a deeper comprehension not only of cloud performance but also of cloud operation and behavior. This is to be achieved through a diverse array of quantitative evaluation tools, including benchmarks, metrics, and workload generators. The workshop places emphasis on exploring novel cloud attributes such as elasticity, performance isolation, dependability, and other non-functional system properties, alongside traditional performance-related metrics such as response time, throughput, scalability, and efficiency.

Following a rigorous review process, HotCloudPerf 2024 featured seven full papers, two short papers and one demo. Furthermore, we are delighted to announce that the workshop will include three keynote talks, delivered by the following speakers.

- Josef Spillner is a senior lecturer/associate professor for computer science at Zurich University of Applied Sciences, Switzerland. His research activity focuses on distributed application computing paradigms. Particular emphasis is on technological support for emerging digitalisation needs of industry and society, such as smart cities and mobility. He will give a talk on the topic: "Upscaling messaging and stateful computation".
- Robert Chatley holds the position of Director of Software Engineering Practice at Imperial College London. His role at Imperial combines a strong focus on education with industry-focused research. Robert has worked with many companies, from startups to multinationals, variously either as a trainer/coach, as a consultant on technical practice, or working as part of engineering leadership. He will give a talk on the topic: "Continuous Developer Feedback for Cloud Native Systems".
- Sasko Ristov is an Assistant Professor for computer science at the University of Innsbruck, Austria. His main research interests include performance modeling and optimization of distributed systems and applications. In particular, he focuses on serverless computing, cloud engineering, and cloud federation. His talk is on the topic: "Engineering serverless application life-cycles in federated serverless infrastructures".

The HotCloudPerf 2024 program committee was composed of the following members: Alexandru Iosup (VU, NL), Nikolas Herbst (U. Würzburg, DE), Cristina Abad (ESPOL, ECU), Auday Al-Dulaimy (Mälardalen University, SE), Andre Bondi (Software Performance and Scalability Consulting LLC, US), Wilhelm Hasselbring (University of Kiel, DE), Dragi Kimovski (University of Klagenfurt, AT), Tania Lorido (Roblox, US), Tiziano De Matteis (VU, NL), Narges Mehran (University of Klagenfurt, AT), Zahra Najafabadi (University of Klagenfurt, AT), Issam Rais (The Arctic University of Norway, NO), Prateek Sharma (Indiana University Bloomington, US), Josef Spillner (ZHAW School of Engineering, CH), Sacheendra Talluri (VU, NL), Klervie Toczé (Linköping University, SE), Petr

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651415

Tůma (Charles University, CZ), André van Hoorn (University of Hamburg, DE), Chen Wang (IBM, US).

We thank all the authors who submitted their research to the workshop and the keynote speakers. We also thank the members of the HotCloudPerf PC for their in depth reviews and discussion. Furthermore, thanks go to the ICPE workshop chairs Diego Costa and Michele Tucci, the ICPE general chairs Simonetta Balsamo and William Knottenbelt, and the complete organization team.

This seventh edition of the HotCloudPerf is supported by the EU Graph-Massiviser project. The HotCloudPerf workshop is technically sponsored by the Standard Performance Evaluation Corporation (SPEC) Research Group (RG), and is organized annually by the RG Cloud Group. HotCloudPerf has emerged from the series of yearly meetings organized by the RG Cloud Group, since 2013. The RG Cloud Group group is taking a broad approach, relevant for both academia and industry, to cloud benchmarking, quantitative evaluation, and experimental analysis.

# Upscaling Messaging and Stateful Computation

Josef Spillner
josef.spillner@zhaw.ch
Zurich University of Applied Sciences
Winterthur, Switzerland

## ABSTRACT

Large-scale, production-grade cloud applications are no longer black boxes for academic researchers. They are observable subjects under test in an increasing number of projects, with the aim to quantify and improve their runtime characteristics, including performance. With more meaningful measurements available, data-driven approaches have matured and advanced the knowledge in particular around conventional stateless workloads such as functions and containers. A few less explored areas still exist. They are fueled by the increasing number of atypical function deployments for instance in message brokers, in intelligent switches and in blockchains. This talk summarises reference architectures for large-scale applications, sometimes resulting in nation-scale deployments, discusses performance numbers in this context, and elaborates on whether more focus on performance is needed.

**ACM Reference Format:**
Josef Spillner. 2024. Upscaling Messaging and Stateful Computation. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3629527.3652885

## 1 INTRODUCTION

Industrially relevant systems engineering has led to high computational complexities, both in pure software systems (e.g. business applications and middleware) and in an increasing variety of systems that depend on combinations of specific hardware and sensors. The complexity is partly driven by the sheer scale, with even small and medium engineering companies often facing the demand to produce, deploy and operate large-scale and even nation-scale applications. With concurrent invocations exceeding the limits of commercial off-the-shelf offerings (e.g. FaaS), engineers are then faced with the tough choice to revert back to self-managed VMs or containers, or to become creative especially with atypical function deployments. This keynote talk encourages to try the latter and to dare looking forward to software and system architectures where the simplicity is maintained and yet the posssible scale is increased significantly. Can entry-level engineers or students be tasked with such a task now? Probably not, but within a few years that may change.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652885

## 2 EXPLORATION

To explore the topic area systematically, a few ideas around the notion of functions must be combined and critically examined. Raw concurrency? Can be achieved with big data processing frameworks, all of which fit into modern cloud hosting by now, but require understanding user-defined functions especially for event-driven (non-batch) stream processing [3], as well as prediction of computation time bases on input sources. Ease of deployment and portability? Second-generation serverless frameworks, running pre-containerised functions, help but come with fixed isolation levels and severe limitations per deployment and per region. In-situ processing of logic within message brokers [4], or even within switches and network interfaces [2]? Brings computation closer to the communication path but often hits limits of devices designed for the latter but not for the former. Blockchains running smart contracts with function semantics? Have recently brought an interesting perspective to stateful functions [1] but, due to the nature of decentralisation, may not be suitable for raw throughput and low latency. Liquid functions across edge-cloud continuums? Prototypes exist...

Given such a large number of choices, systematic evaluations and practical experience are both required to come up with useful advice to engineers in the form of checklists, patterns, metaprogramming and other simplifications at the textbook level. The challenges are not only in the infrastructure with hardware constraints and arbitrarily set limits, but deeply routed in the messaging protocols (e.g. privacy considerations) and compute units. With reference to currently ongoing industry-funded research and innovation projects, the talk therefore goes beyond the encouragement and demonstrates the practical need to find convincing system designs, under the hypothesis that atypical function deployments (beyond mainstream FaaS) plays a role in this search process. It gives examples from several cyber-physical application domains such as monitoring people flows and herd movement, as well as digital pandemic and academic credentials management.

## REFERENCES

[1] Maksym Arutyunyan, Andriy Berestovskyy, Adam Bratschi-Kaye, Ulan Degenbaev, Manu Drijvers, Islam El-Ashi, Stefan Kaestle, Roman Kashitsyn, Maciej Kot, Yvonne-Anne Pignolet, Rostislav Rumenov, Dimitris Sarlis, Alin Sinpalean, Alexandru Uta, Bogdan Warinschi, and Alexandra Zapuc. 2023. Decentralized and Stateful Serverless Computing on the Internet Computer Blockchain. In *2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*, Julia Lawall and Dan Williams (Eds.). USENIX Association, 329–343.

[2] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. 2021. Speedo: Fast dispatch and orchestration of serverless workflows. In *SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021*, Carlo Curino, Georgia Koutrika, and Ravi Netravali (Eds.). ACM, 585–599. https://doi.org/10.1145/3472883.3486982

[3] Yannis Foufoulas and Alkis Simitsis. 2023. Efficient Execution of User-Defined Functions in SQL Queries. *Proc. VLDB Endow.* 16, 12 (aug 2023), 3874–3877. https://doi.org/10.14778/3611540.3611574

[4] K. Sundar Rajan, A. Vishal, and Chitra Babu. 2021. A Scalable Data Pipeline for Realtime Geofencing Using Apache Pulsar. In *Comp. Intellig. in Data Science - 4th IFIP TC 12 Intl. Conf., ICCIDS 2021, Chennai, India, March 18-20, 2021 (IFIP Advances in Information and Communication Technology, Vol. 611)*, Vallidevi Krishnamurthy, Suresh Jaganathan, Kanchana Rajaram, and Saraswathi Shunmuganathan (Eds.). Springer, 3–14. https://doi.org/10.1007/978-3-030-92600-7_1

# Engineering Serverless Application Life-cycles in Federated Serverless Infrastructures

Sashko Ristov
sashko.ristov@uibk.ac.at
University of Innsbruck
Department of Computer Science
Innsbruck, Tyrol, Austria

## ABSTRACT

The top cloud providers offer more than a hundred serverless services, such as Function-as-a-Service and various ML-based Services speech to text, text to speech, or translation. Unfortunately, while the cloud provider SDKs simplify the usage of serverless services, they also lock the users to use services of the respective provider only. Moreover, the dynamic and heterogeneous nature of the underlying serverless infrastructure introduces other deficiencies for agile development, automated deployment, and efficient and effective execution of serverless workflow applications.

This talk will present our advances [1, 2] in many steps of serverless application life-cycles: development, modeling, and running serverless workflow applications that use various serverless managed services in federated serverless infrastructures. The main goal is to follow the approach "Code Once Run Everywhere" where the developers code their "intents" and the runtime systems then selects the specific deployment of end-point managed cloud services.

## CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

**ACM Reference Format:**
Sashko Ristov. 2024. Engineering Serverless Application Life-cycles in Federated Serverless Infrastructures. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3629527.3652886

## AUTHOR KEYWORDS

interoperability, optimization, performance, serverless, simulation, workflow applications.

## BIOGRAPHY

Sashko Ristov is Assistant Professor for computer science at the University of Innsbruck, Austria. His main research interests include performance modeling and optimization of distributed systems and applications. In particular, Dr. Ristov focuses on serverless computing, cloud engineering, and cloud federation.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652886

## ACKNOWLEDGEMENT

This research received funding from Land Tirol, under the contract F.35499.

## REFERENCES

[1] . 2024. FaaS Tools. Retrieved 2024-03-17 from https://github.com/FaaSTools/
[2] . 2024. xAFCL Serverless Workflow Management System. Retrieved 2024-03-17 from https://github.com/xAFCL

# A Systematic Configuration Space Exploration of the Linux *Kyber* I/O Scheduler

Zebin Ren
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Krijn Doekemeijer
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Animesh Trivedi
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

## ABSTRACT

NVMe SSDs have become the de-facto storage choice for high-performance I/O-intensive workloads. Often, these workloads are run in a shared setting, such as in multi-tenant clouds where they share access to fast NVMe storage. In such a shared setting, ensuring quality of service among competing workloads can be challenging. To offer performance differentiation to I/O requests, various SSD-optimized I/O schedulers have been designed. However, many of them are either not publicly available or are yet to be proven in a production setting. Among the widely-tested I/O schedulers available in the Linux kernel, it has been shown that *Kyber* is one of the best-fit schedulers for SSDs due to its low CPU overheads and high scalability. However, *Kyber* has various configuration options, and there is limited knowledge on how to configure *Kyber* to improve applications' performance. In this paper, we systematically characterize how *Kyber*'s configurations affect the performance of I/O workloads and how this effect differs with different file systems and storage devices. We report 11 observations and make 5 guidelines that indicate that (i) *Kyber* can deliver up to 26.3% lower read latency than the *None* scheduler with interfering write workloads; (ii) with a file system, *Kyber* can be configured to deliver up to 35.9% lower read latency at the cost of 34.5%–50.3% lower write throughput, allowing users to make a trade-off between read latency and write throughput; and (iii) *Kyber* leads to performance losses when *Kyber* is used with multiple throughput-bound workloads and the SSDs is not the bottleneck. Our benchmarking scripts and results are open-sourced and available at: https://github.com/stonet-research/hotcloudperf24-kyber-artifact-public.

## CCS CONCEPTS

• **Software and its engineering** → **Secondary storage**; *Operating systems.*

## KEYWORDS

Linux storage schedulers, Kyber, Measurements

**ACM Reference Format:**
Zebin Ren, Krijn Doekemeijer, and Animesh Trivedi. 2024. A Systematic Configuration Space Exploration of the Linux *Kyber* I/O Scheduler. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651416

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651416

## 1 INTRODUCTION

Modern high-performance solid-state drives (SSDs) are able to deliver millions of I/O operations per second (IOPS) with single-digit microsecond-level latency [5, 11, 12]. These devices are widely used in multi-tenant cloud environments for their improved performance over hard disks [30, 36, 46]. Cloud providers need to provide quality of service (QoS) guarantees for I/O services such as throughput or tail latency service-level objectives across multiple tenants. These guarantees are usually achieved by scheduling I/O requests with an I/O scheduler [29, 34].

However, existing Linux I/O schedulers designed for hard disks do not work well with these high-performance SSDs and induce significant CPU and scalability overheads [38, 42]. To reduce these overheads, there are many *state-of-the-art* I/O schedulers designed for SSDs [21, 24, 26, 27, 31–33, 35, 39, 41, 43]. Despite these studies on I/O schedulers for SSDs, using these past published I/O schedulers is challenging. Many of them do not have their source code public or are written for a specific kernel version, or assume specific hardware support from SSDs [24, 43]. Thus, users need to implement these I/O schedulers in the Linux kernel, which is not trivial, preventing their widespread use.

Compared to these *state-of-the-art* I/O schedulers, the *state-of-the-practice* plug-and-play Linux I/O schedulers [7], *Kyber* [6], *MQ-Deadline* [8], and *BFQ* [2], are the most accessible schedulers. In our past studies, we demonstrate that *Kyber* has a low CPU overhead and high scalability on fast SSDs and recommend using *Kyber* on high-performance SSDs for its low CPU overhead and high scalability [37, 38]. We also identify that *Kyber*'s configuration significantly impacts workload performance in terms of latency and throughput, and this impact also differs between different workloads [37]. *Kyber* provides two configurable parameters, read and write target latency, allowing users to set the target latencies that *Kyber* should try to deliver. The effect of *Kyber*'s configurations and the difference of this effect on different workloads create challenges in using *Kyber* in practice. There is no existing study on configuring *Kyber* for specific software and hardware settings. Specifically, how to find an optimized *Kyber* configuration with a specific setting for different (1) workloads, (2) file systems, and (3) types of SSDs.

In our study, we cover these three aspects to show the effect of *Kyber*'s configurations on its performance. Firstly, workloads have different I/O patterns and latency/throughput requirements [23, 24]. Existing studies of *Kyber* focus on its CPU and latency overhead, scalability, and its ability to deliver low latency for foreground workloads [23, 33, 38, 42]. There is a lack of systematic studies on how *Kyber*'s configurations affect the performance of interfering concurrent workloads with diverse demands in terms of expected read/write latencies and throughputs. Moreover, predicting the achieved performance with the latency targets is not trivial

since the user-specified latency targets are not guaranteed by *Kyber*. Thus, there is a gap between the target performance (*Kyber*'s configurable parameters of read/write latencies) and the achieved performance (latency and throughput), which is not obvious from the latency targets configured. Secondly, real-world workloads usually work with file systems instead of directly accessing the storage device. File systems change the I/O patterns of the workloads. Thus, the effect of *Kyber* on workload performance with different file systems is unknown. Thirdly, different types of SSDs have significantly different performance properties such as peak throughput, latency, and read/write interference behavior [23, 35]. For example, flash-based SSDs have unpredictable performance and read/write interference, but non-flash-based ultra-low latency (UUL) SSDs such as Intel Optane SSDs have stable performance and no read/write interference [44, 47].

In conclusion, the lack of understanding of how *Kyber*'s configurations affect the achieved performance with different workloads, file systems, and types of SSDs makes it unclear how to optimize *Kyber* in practice. Specifically, we investigate the following research questions (**RQs**) around how *Kyber*'s configurations affect the workloads' performance with different workloads, file systems and types of SSDs:

**(RQ1) How does *Kyber* affect the performance of workloads when workloads run concurrently and interfere with each other?** We investigate how *Kyber* affects the performance of different workloads by studying the relation between target latency and the workloads' achieved performance.

**(RQ2) How to configure *Kyber*'s parameters for diverse types NVMe SSDs and diverse file systems to meet workloads' requirements?** The key motivation is to find out if and how our findings on *Kyber*'s configurations performance effects can be generalized to different file systems and types of SSDs. We also provide guidelines on how to configure *Kyber* to meet the workloads' requirements in practice with diverse software and hardware environments.

To address these questions, we conduct a first-of-its-kind systematic study of Linux' *Kyber* I/O scheduler with various kinds of workloads, file systems, and types of SSDs to establish guidelines on how to configure *Kyber* in practice. Our key contributions in this work include:

- We extensively study how *Kyber* with different configurations affects workload performance using different combinations of latency-sensitive and throughput-bound workloads on 2 types of SSDs, resulting in 11 observations. To the best of our knowledge, we are the first to investigate the effect of *Kyber*'s configurations on workloads.
- Based on our observations, we provide 5 guidelines on how to configure *Kyber* in practice with various workloads, file systems, and types of SSDs.
- We open-source all artifacts, datasets, and scripts for this paper as FAIR data sets at https://github.com/stonet-research/hotcloudperf24-kyber-artifact-public.

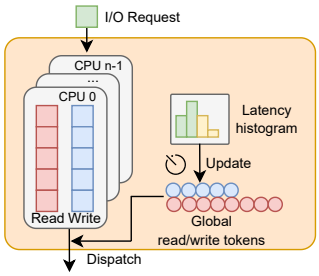



**Figure 1: Architecture of the Linux *Kyber* I/O scheduler.**

## 2 BACKGROUND

**NVMe SSDs.** Non-volatile memory express (NVMe) is an interface for accessing storage devices through PCIe. NVMe is widely used by high-performance SSDs. In this paper, we evaluate two kinds of NVMe SSDs: flash-based SSDs and non-flash-based SSDs with the 3D Xpoint technology [1].

Flash-based SSDs are composed of a controller that is connected to an array of flash chips. Each flash chip is organized in a hierarchy of dies, planes, blocks, and pages. SSDs have high internal parallelism as both dies and planes can operate in parallel. The NVMe protocol [10] exposes this parallelism to workloads with a multi-queue interface that allows SSDs to execute multiple I/O requests in parallel. Nevertheless, to fully utilize this parallelism, workloads need to issue multiple concurrent I/O requests to the SSD. A challenge here is that a plane can not execute different types of commands (read or write) in parallel. If a read is issued to a die where a write is already being executed, the read is blocked until the write finishes, leading to a 10–40× longer read latency. This performance degradation is called read/write interference [17, 45]. Moreover, the physical constraints of flash chips do not allow in-place updates or intra-block random writes. Pages in a block can only be written sequentially, and written pages need to be erased before they can be rewritten. Erasures happen at the unit of blocks, not at the unit of pages. To imitate the block interface provided by hard disks, the Flash Translation Layer (FTL) in SSD controllers maps logical addresses provided in the block interface to physical addresses in the flash chips. On an update, the data of the update is written to a new page, and the old page is marked invalid. The internal operations lead to additional interference with user I/O requests, leading to unpredictable performance. In conclusion, flash-based SSDs have (1) high parallelism, (2) unpredictable performance, and (3) read/write interference.

There are also non-flash-based SSDs such as Intel Optane SSDs [4], made with 3D Xpoint technology [1, 44]. 3D Xpoint has two big differences from flash: (1) it is byte-addressable, thus an I/O request can be broken into smaller pieces and processed in parallel by multiple channels to achieve low latency; and (2) it supports in-place updates and can, thus, provide stable performance without internal translation operations needed such as flash-based SSDs [47].

***Kyber* Internals.** *Kyber* is an I/O scheduler designed for fast and highly parallel storage devices inspired by active queue management techniques from network routing [6, 13]. *Kyber* prioritizes reads over writes based on the heuristic that a process that issues a read request usually waits for the issued read to finish. In contrast, a process that issues a write request usually continues executing

**Table 1: Benchmarking environment.**

| *Component* | *Configuration* |
|---|---|
| CPU | Single socket Intel(R) Xeon(R) Silver 4210R CPU 10 cores @ 2.40GHz, Hyper-threading disabled, Turbo disabled. |
| Memory | 256 GiB, DDR4. |
| Storage | Samsung 980 PRO 1 TiB (Flash-based SSD); Intel SSD 900P (Optane SSD) |
| Software | Ubuntu 20.04 with Linux kernel `v6.3.8`, `fio` v3.35. |

without waiting for the write to finish. Figure 1 shows the architecture of the Linux *Kyber* I/O scheduler. *Kyber* maintains two queues for each CPU core, one for reads and one for writes. *Kyber* inserts I/O requests into the queues on the same core where the application issued the requests. These read/write queues are associated with a global token bucket. These tokens are used to limit the number of concurrent requests issued to the SSD to achieve high responsiveness. An I/O request is dispatched to the NVMe device driver only when there are available tokens. The number of tokens remains the same if both read and write target latencies are satisfied, and is increased if read or write P99 latency exceeds the target latency. Increasing the number of tokens increases the priority of that workload. The number of NVMe tokens for a particular type of request (read or write) is reduced when (1) the achieved P90 latency for that request type is lower than the target latency; and (2) the achieved P99 latency for the other type is higher than the target latency. *Kyber* aims to deliver the user configured target latencies. However, there is no guarantee that *Kyber* achieves the target latencies. Further on, it is not studied how these target latencies affect the achieved workload throughput and latency. The default read and write target latencies are 2 ms and 10 ms, respectively. However, achievable latencies for NVMe SSDs range from 10 to 80 $\mu$s and differ between different types of SSDs [4, 11]. Therefore, there is a huge gap between the default target latencies and the best latencies that NVMe SSDs can deliver. It is unknown how this gap and the performance difference of SSDs affect workload performance on NVMe SSDs when using *Kyber*. The aim of this paper is to investigate how *Kyber*'s target latencies and the performance of different SSDs affect the achieved workload throughput and latency.

## 3 METHODOLOGY

**Hardware and Software.** Our benchmarking environment is shown in Table 1. We use `fio` [3] as a workload generator with the io_uring interface [14]. All the I/O requests are issued with the `O_DIRECT` flag so the I/O requests bypass the page cache. We use two metrics to evaluate performance: throughput and latency. We measure throughput in I/O operations per second (IOPS) and latency in 99 percentile operation tail latencies (P99 latency). Before running the experiments, we precondition the flash SSD according to [16]—by sequentially writing the entire SSD, then writing 2 TiB of 4 KiB random writes. We run each experiment on the Samsung 980 PRO for 12 minutes (6 minutes warm-up time + 6 minutes run time) with five repetitions. For each experiment, we report the average(s) of these five runs. On the Intel Optane SSD, we run each experiment with one repetition for 2 minutes and 30 seconds (30 seconds warm-up time + 2 minutes run time). We use a shorter run time for the experiments on the Optane SSD because it delivers stable performance.

**Synthetic Workloads and Methodology.** Workloads in cloud environments have diverse I/O requirements, such as latency-sensitive workloads (e.g., online database query) and throughput-bound workloads (e.g., batch processing systems) [24]. We use two synthetic workloads: latency-sensitive workloads (L-app) and throughput-bound workloads (T-app). Both only issue 4 KiB read or write requests. For the L-apps, we issue a single outstanding request (we use queue depth, or QD to represent 'the number of outstanding requests' for simplicity in later sections). The T-apps issue 256 outstanding requests to saturate the SSDs. In the following sections, we use `R1` and `W1` to represent L-app read and write workloads respectively and `R256` and `W256` to represent T-app read and write workloads respectively. The number after `R` and `W` represents the QD of the workload.

**Table 2: Baseline performance of Samsung 980 PRO SSD with the *None* scheduler.**

| # | Workload(s) | R TP (in KIOPS) | W TP (in KIOPS) | R P99 Lat (in $\mu$s) | W P99 Lat (in $\mu$s) |
|---|---|---|---|---|---|
| 1 | `R1` | 17.0 | - | 77.5 | - |
| 2 | `R256` | 364.3 | - | 793.8 | - |
| 3 | `W1` | - | 62.3 | - | 23.1 |
| 4 | `W256` | - | 70.0 | - | 15,794.2 |
| 5 | `R1-W1` | 4.0 | 65.0 | 1,879.2 | 26.8 |
| 6 | `R1-W256` | 0.3 | 68.9 | 15,217.5 | 15,558.2 |
| 7 | `R256-W1` | 302.6 | 61.5 | 3,044.1 | 32.1 |
| 8 | `R256-W256` | 83.2 | 93.1 | 15,283.0 | 15,938.4 |

## 4 BASELINE PERFORMANCE WITH THE *NONE* SCHEDULER

As we explained in §2, flash-based SSDs have read/write interference, which means that a write blocks concurrent reads to the same die. In this section, we establish the baseline performance of the evaluated flash SSDs with and without interference. We report the read/write throughput and latency with different workload combinations (i.e., L-app, T-app). We use the *None* scheduler, a no-op scheduler, which passes the I/O requests to the NVMe device driver in a first-in-first-out manner. Each workload is pinned to a dedicated CPU core to avoid interference by the process scheduler. Table 2 shows the throughput (in KIOPS) and P99 tail latency (in $\mu$s) of different workload combinations. We show the (combination) of workloads in the second column and we show the throughput and latency of the read and write workloads in the third to sixth columns. We have three observations:

**Asymmetric read/write performance.** The flash SSDs have asymmetric read/write performance **(Observation 1, O-1)**. With a single CPU core and no interfering workloads, the flash SSD delivers up to 364.3 KIOPS random read throughput at QD=256 and 77.5 $\mu$s P99 random read latency at QD=1 (row 1). When `fio` issues random writes, the flash SSD delivers up to 70.0 KIOPS throughput at QD=256 (row 4) and 23.1 $\mu$s P99 latency at QD=1 (row 3). In short, the flash SSD has different throughput and latency for reads and writes without interference. Next, we show how this performance changes with the interference of a second workload.

**Writes have a huge impact on read performance.** A concurrent write workload significantly degrades the performance of a co-running read workload. When a latency-sensitive read workload `R1` is mixed with a latency-sensitive write workload `W1` (row 5), the read throughput drops 76.5% (from 17.0 to 4.0 KIOPS) and the latency increases 24.2× (from 77.5 to 1,879.2 $\mu$s) compared to `R1` without interference (row 1). The read performance degradation is more significant with a throughput-bound write workload `W256` (row 6), showing 98.2% lower throughput (from 17.0 to

**Table 3: Evaluated workloads combinations with *Kyber*.**

| | L-app with write (W1) | T-app with write (W256) |
|---|---|---|
| L-app with read (R1) | R1–W1 | R1–W256 |
| T-app with read (R256) | R256–W1 | R256-W256 |

0.3 KIOPS) and 200.8× higher latency (from 77.5 to 15,217.5 $\mu$s) than R1 without interference. When a read throughput-bound workload and a write throughput-bound workload compete for throughput (row 8), the read workload has 77.2% lower throughput (from 364.3 to 83.2 KIOPS) than without the interference of concurrent writes. To conclude, the read performance is highly sensitive to the write workload, changing the write workload leads to a significant effect on read throughput and latency **(O-2)**.

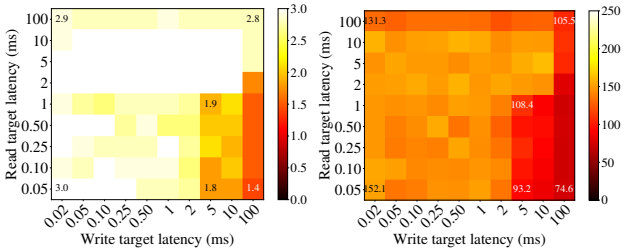
**Reads have a less significant impact on write performance.** A co-running read workload has less impact on the write performance than the impact write workloads have on read workloads. When W1 runs with R1 in the background (row 5), the write workload has comparable throughput (from 62.3 to 65.0 KIOPS), and the latency only increases by 16.0% (23.1 to 26.8 $\mu$s) compared to running W1 in isolation. With a throughput-bound workload R256 on the background (row 7), the write latency increases 39.0% (from 23.1 to 32.1 $\mu$s), much lower than the latency increase of reads in this setting (200.8×). Thus, the write performance is less sensitive to the read workload than the interference of writes on reads **(O-3)**.

The key finding here is that the *None* scheduler can not mitigate read/write interference. When a latency-sensitive read workload runs concurrently with a write workload, the read workload has a significantly higher P99 tail latency than the read workload running in isolation. When there are two throughput-bound read and write workloads competing for throughput, *None* does not provide any functionality to tune the throughput share between the read and write workloads. *Kyber* offers configuration options that let the users prioritize reads or writes over each other. Thus, in the following sections, we investigate if and how *Kyber* affects read/write interference and how it affects the throughput share between throughput-bound read and write workloads under different configurations.

We repeat the same benchmark on an Intel Optane P900 SSD (the results are not plotted in the paper). We have two observations. Firstly, the Optane SSD have symmetric read/write performance. Unlike the flash SSD, the Optane SSD deliver comparable throughput and latency for both reads and writes. Secondly, the Optane SSD have less read/write interference. R1 has up to 65.6% higher latency (15.4 vs. 44.8 $\mu$s) with a concurrently running W256 compared to running R1 in isolation, which is significantly lower than the Samsung 980 PRO (24.2× lower). We show how *Kyber* and it's configurations affect the performance of these workloads in §5.

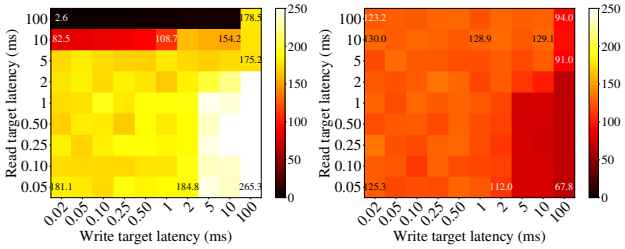
## 5 PERFORMANCE EFFECT OF *KYBER*'S CONFIGURATIONS WITHOUT A FILE SYSTEM

We start our analysis with a performance characterization of the impact of different *Kyber* configurations on fio-based micro-benchmarks. We run these micro-benchmarks without any file system. Specifically, we investigate how different *Kyber* configurations affect the P99 latency of L-apps and throughput of T-apps when concurrent




**(a) R1 P99 latency (in ms)** **(b) W256 throughput (in KIOPS)**

**Figure 2: Performance of L-app (read-only) and T-app (write-only) workload combinations with different *Kyber* configurations.**




**(a) R256 throughput (in KIOPS)** **(b) W256 throughput (in KIOPS)**

**Figure 3: Performance of T-app read and write workload combinations with different *Kyber* configurations.**

read and write workloads interfere with each other, see Table 3 for all combinations. Such a setup is common in the multi-tenant cloud. For each benchmark, we start two concurrent fio processes. One fio process issues reads and one process issues writes. Each fio process is pinned to a separate and dedicated CPU core to prevent them from competing for CPU resources. We do a grid search to investigate how *Kyber*'s configurations affect the achieved performance of fio workloads in the search space. We set the lowest read and write target latency to 50 $\mu$s and 20 $\mu$s respectively, based on the minimum P99 latency of the flash SSD (§4), and we gradually increase the target latency to 100 ms. We report our performance results in Figure 2 and Figure 3 as heatmaps where the x-axis represents the write target latency and the y-axis represents the read target latency. The temperature in the heatmaps is the measured performance with read and write target latencies set to corresponding values in y- and x-axis.

**How does *Kyber* affect the performance of different combinations of workloads?** We report that *Kyber*'s configurations do not have a significant effect on the performance of workload combinations R1–W1 (thus they are not plotted in the paper) **(O-4)**. We do not observe a relation between the P99 read/write latencies and *Kyber* configurations. The P99 read latency varies between 1.3 and 1.6 ms and the P99 write latency varies between 23.4 and 27.3 $\mu$s. The reason that *Kyber* does not have a significant effect on R1–W1 is that *Kyber*'s mechanism is only effective with multiple outstanding I/O requests. Yet, with R1–W1, there is only 1 outstanding read and write request. Since *Kyber* allows at least one read and one write to be sent to the SSDs, thus *Kyber* does not throttle the request with R1–W1. We report that *Kyber* is effective when there is at least

more than one outstanding read or more than one outstanding write **(Guideline 1, G-1)**.

**Can *Kyber* provide bounded P99 latency for the L-app when an L-app interferes with a T-app?** Figure 2 shows how *Kyber*'s configuration affects `fio`-workload performance when a read L-app (`R1`) and a write T-app (`W256`) run concurrently. The temperature in Figure 2a shows the read P99 latency (in ms, darker is better) of the read L-app and the temperature in Figure 2b shows the throughput (in KIOPS, lighter is better) of the write T-app. In our experiments, *Kyber* mitigates read/write interference at the cost of write throughput **(O-5)**. When the read target latency is set to 50 $\mu$s and the write target latency is set to 100 ms, the achieved read P99 latency is 1.4 ms, 26.3% lower (1.8 ms) than the read latency of `R1–W1` with the *None* scheduler (row 5, Table 2). Thus, *Kyber* delivers low read latency with background throughput-bound write workloads by setting the read target latency to the lowest read P99 latency that the SSD can achieve (50 $\mu$s in our case) and the write target latency to a value higher than the achieved write latency with the *None* scheduler (15.6 ms, row 6, Table 3). However, the cost of achieving low read latency is lower write throughput (from peak 152.1 to 74.6 KIOPS, 50.9% lower throughput). We suggest using *Kyber* in multi-tenant situations when low read latency is considered more important than high throughput **(G-2)**.

**How do *Kyber*'s configurations affect the throughput share of two throughput-bound `fio`-workloads?** Figure 3 shows how *Kyber*'s configurations affect the interference between a read T-app and a write T-app. The temperature in Figure 3a shows the read throughput and the temperature in Figure 3b shows the write throughput (in KIOPS, lighter is better). Firstly, decreasing the read target latency leads to higher read throughput. With a fixed write target latency (fixed x value), as the read target latency decreases, the read throughput increases **(O-6)**. For example, with the write target latency is set to 20 $\mu$s (first column in Figure 3a), as the read target latency decreases from 100 ms to 50 $\mu$s, the read throughput increases from 2.6 KIOPS to 181.1 KIOPS, a 69.7× increase. Secondly, when the read target latency is lower than 10 ms and the write target latency is lower than 5 ms, changing *Kyber* configurations does not lead to a statistical difference in read and write throughput. The reason is that the achieved read and write latencies are 5 ms and 18 ms (not visualized). Tuning the target latencies in a configuration space where all the candidate values are much lower than the lowest achievable latency, all the configurations in this configuration space lead to comparable performance (we call this space the *dead configuration space*). In conclusion, by tuning *Kyber*'s configuration, the throughput between reads and writes can be distributed. We suggest that the users (1) run this grid search micro-benchmark in Figure 2 and Figure 3 to find out how the target latencies affect the performance of a specific SSD and (2) avoid tuning *Kyber* in the dead configuration space **(G-3)**.

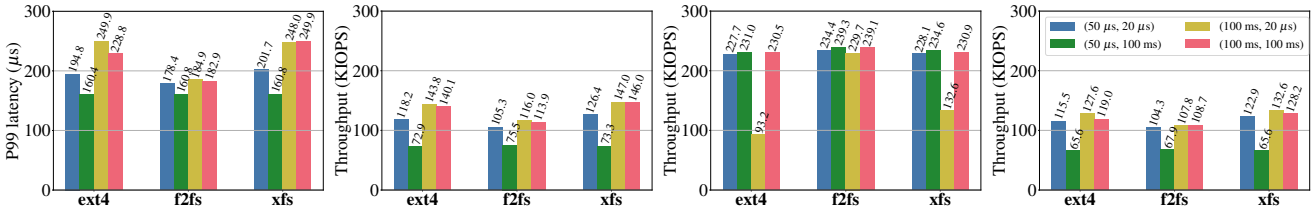
**How does this effect change with different SSDs?** We repeat our experiments on the Intel Optane SSD to investigate how this effect varies across different SSDs. Firstly, similar to the Samsung SSD, we observe that with `R1–W256`, prioritizing reads by setting low read target latency and high write target latency leads to lower P99 read latency at the cost of write throughput. When *Kyber* is configured to prioritize reads, it delivers 62.5% lower latency (from 44.8 to 16.8 $\mu$s) than it does with the *None* scheduler at the cost of 67.1% lower write throughput (from 228.2 to 75.0 KIOPS) **(O-7)**. Secondly, when two throughput-bound workloads interfere with each other (`R256–W256`), we report that prioritizing reads leads to lower write throughput (from 202.5 to 68.9 KIOPS) and comparable read throughput when prioritizing writes. However, when neither reads nor writes are prioritized, setting the read and write latency to the same value leads to high read and write throughputs (215.4 and 203.1 KIOPS, respectively) at the same time. The explanation for this phenomenon is that the Optane SSD has low read/write interference [44]. When the SSD is not saturated, adding a concurrent write workload with a read workload does not have a significant effect on the performance of the read workload. In this setting (`R256–W256`), the SSD is not saturated. In short, limiting read (or write) throughput does not increase the write (or read) throughput on the Optane SSD. With the Optane SSD, a misconfiguration when the SSD is not saturated leads to a throughput drop for reads or writes without any throughput increase for writes or reads **(O-8)**. Thus, we suggest using *Kyber* with Optane SSDs only when the SSDs are the bottleneck.

## 6 PERFORMANCE EFFECT OF *KYBER*'S CONFIGURATIONS WITH FILE SYSTEMS

In the previous section, we investigate how *Kyber* affects the performance of `fio`-workloads without using any file system. However, real-world workloads usually access SSDs via file systems. In this section, we characterize how *Kyber*'s configuration affects the I/O performance with three different file systems: ext4 [9], f2fs [28], and xfs [22]. The goal is to investigate if the observations of our microbenchmarks (§5) generalize to file systems. We evaluate two workload combinations: `R1–W256` and `R256–W256` in Table 3 with four *Kyber* configurations where the target read and write latency is set to (50 $\mu$s R, 20 $\mu$s W), (50 $\mu$s R, 100 ms W), (100 ms R, 20 $\mu$s W) and (100 ms R, 100 ms W), the four extreme configurations in the configuration search space in the previous section. The performance of the `fio` workloads is reported in Figure 4.

**How does *Kyber* affect the I/O performance with the use of a file system?** We first investigate how *Kyber*'s configurations affect the performance of `R1–W256` with different file systems. Figure 4a and Figure 4b show the P99 read latency (in $\mu$s, the lower the better) and write throughput (in KIOPS, the higher the better) respectively with workload `R1–W256`. *Kyber* delivers the lowest read P99 latency with configuration (50 $\mu$s R, 100 ms W), 13.0%–35.9% lower latency than the worst configuration (160.4–160.8 $\mu$s vs. 184.9–249.9 $\mu$s) . This lower P99 read latency comes at the cot of lower write throughput (72.9–75.5 KIOPS or 34.9%–50.1% lower) compared to the highest write throughput (116.0–147.0 KIOPS) **(O-9)**. If the workloads access the SSD via a file system, *Kyber* can be configured to deliver up to 35.9% lower read P99 latency than the P99 read latency delivered in other configurations with concurrent background writes **(G-4)**.

Next, we investigate how *Kyber*'s configurations affect the throughput when a read throughput-bound workload and a write throughput-bound workload run concurrently. Figure 4c and Figure 4d show the read and write throughput with workload setting `R256–W256`. We have two observations. Firstly, the workloads with ext4 and xfs have similar performance. Configuring *Kyber* to prioritize read (e.g.,




(a) R latency in R1–W256 (b) W throughput in R1–W256 (c) R throughput in R256–W256 (d) W throughput in R256–W256

**Figure 4: Performance of R1–W256 and R256–W256 combinations with different *Kyber* configurations and ext2, f2fs and xfs.**

50 $\mu$s read latency and 100 ms write latency, the second bar in each group) does not lead to significantly higher read throughput compared to the other three settings (from 227.7 and 228.1 KIOPS to 231.1 and 234.6 KIOPS, 1.3% and 2.8% higher read throughput). However, the write throughput is significantly decreased (from 127.6 and 132.6 KIOPS to 65.6 KIOPS, 48.6% and 50.5% lower throughput). The same occurs when we configure *Kyber* to prioritize writes over reads (e.g., 100 ms write latencies and 20 $\mu$s read latencies, the third bar in each group) **(O-10)**. Secondly, *Kyber*'s configurations do not have a significant effect on the read throughput of f2fs (the read throughput is 229.7–239.3 KIOPS). However, prioritizing reads causes the write throughput to decrease from 108.7 KIOPS to 67.9 KIOPS, a 37.5% lower write throughput **(O-11)**. Thus, we recommend that users should configure *Kyber* with the same read and write target latencies. In our setup, the read and write target latencies are set to (50 $\mu$s R, 20 $\mu$s W) and (100 ms R, 100 ms W) to achieve both read and write peak throughput **(G-5)**.

In conclusion, when a latency-sensitive workload runs concurrently with a throughput-bound write workload via a file system, *Kyber* can be configured to deliver low read P99 latency by setting low read target latency and high write target latency to prioritize reads. When there are read and write throughput workloads running concurrently, we suggest setting the read and write target latencies to similar values to achieve high read and write throughput.

## 7 RELATED WORK

**I/O schedulers for flash SSDs.** Our study focuses on the *state-of-the-practice* Linux I/O scheduler *Kyber*. However, there are many *start-of-the-art* I/O schedulers for SSDs for Linux.

Designing fair-sharing I/O schedulers has been extensively studied with SSDs [15, 18, 21, 35, 39, 40, 43, 48]. MQFQ [21] utilizes the multi-queue interface to increase its scalability. D2FQ [43] further increases the performance of fair-sharing I/O schedulers by eliminating the "stage" step and offloading the scheduling to SSDs using the weighted round-robin feature [25].

There are also I/O schedulers that are optimized to deliver low latency for latency-sensitive workloads in shared environments [24, 31, 33]. K2 [33] strictly prioritizes high-priority requests and trades throughput for latency. blk-switch [24] provides low latency for high-priority workloads and preserves high total throughput at the same time. FastResponse [31] co-designs the I/O scheduler with the storage stack to reduce the I/O interference.

Flash-based SSDs have many idiosyncrasies because of their complex internal architectures. Various I/O schedulers are built to utilize these idiosyncrasies to increase SSDs' write performance and lifespan by using fine-grained access [41], reducing SSD GC overhead [19, 20, 26] and reducing read/write interference [27, 35].

**Performance characterization of Linux I/O schedulers.** Many studies characterize the performance of Linux I/O schedulers with NVMe SSDs [37, 38, 42]. Whitaker et al. [42] characterize the performance of the Linux I/O schedulers on ULL SSDs based on 3D XPoint technology. Their findings include that Linux I/O schedulers lead to higher latency, lower throughputs, and higher energy overhead than without the I/O schedulers. Ren et al. [37] extended this work by characterizing the performance overhead, scalability, QoS with more common flash-based SSDs. Additionally, they characterize how *Kyber*'s configurations affect the interference between foreground read workloads and background write workloads. We extend this work on *Kyber* by characterizing the performance of *Kyber* with different combinations of workloads and how these effects can be generalized to different file systems. We presented an in-depth, systematic study to give guidelines on how to configure *Kyber* with specified SSDs and workloads.

## 8 CONCLUSION AND FUTURE WORK

In this paper, we investigate how *Kyber*'s configurations affect the performance of different workloads with various file systems and storage devices. Our results show that *Kyber* can be configured to deliver low read latency when there is a concurrently running write workload. *Kyber* can also be used to balance the throughput share between read and write throughput-bound workloads when the applications directly run on the top of block devices.

This work can be expended in (1) evaluating how *Kyber*'s configuration affects the performance of applications with mixed read and write workloads, (2) designing an automatic tool that can find the best *Kyber* configuration automatically, and (3) designing algorithms that dynamically configures *Kyber* when the workload changes.

**Acknowledgments** This work is partially supported by Netherlands-funded projects from the Dutch Research Council (NWO) grants (OCENW.KLEIN.561 and OCENW.KLEIN.209) and GFP 6G FNS, and EU-funded projects MCSA-RISE Cloudstars and Horizon Graph-Massivizer. Krijn Doekemeijer is funded by the VU PhD innovation program. We thank the anonymous HotCloudPerf'24 reviewers for their invaluable and constructive feedback. We would also like to thank Jesse Donkervliet, Sacheendra Talluri, Matthijs Jansen, and the AtLarge group at VU Amsterdam for their help with the paper.

# REFERENCES


[1] Accessed: 2024-03-13. 3D XPoint. https://insidehpc.com/2015/07/intel-and-micron-announce-3d-xpoint-non-volatile-memory/

[2] Accessed: 2024-03-13. BFQ Budget Fair Queueing Document. https://www.kernel.org/doc/html/latest/block/bfq-iosched.html

[3] Accessed: 2024-03-13. fio. https://github.com/axboe/fio

[4] Accessed: 2024-03-13. Intel Optane 900P Techinical Specification. https://www.intel.com/content/www/us/en/products/sku/123623/intel-optane-ssd-900p-series-280gb-2-5in-pcie-x4-20nm-3d-xpoint/specifications.html

[5] Accessed: 2024-03-13. Intel® Optane™ SSD DC P5800X Series. https://ark.intel.com/content/www/us/en/ark/products/201859/intel-optane-ssd-dc-p5800x-series-1-6tb-2-5in-pcie-x4-3d-xpoint.html

[6] Accessed: 2024-03-13. Kyber Multiqueue I/O Scheduler. https://lwn.net/Articles/720071/

[7] Accessed: 2024-03-13. Linux I/O Schedulers. https://wiki.ubuntu.com/Kernel/Reference/IOSchedulers

[8] Accessed: 2024-03-13. MQ-Deadline Implementation. https://elixir.bootlin.com/linux/latest/source/block/mq-deadline.c

[9] Accessed: 2024-03-13. The New ext4 Filesystem: Current Status and Future Plans. https://www.kernel.org/doc/ols/2007/ols2007v2-pages-21-34.pdf

[10] Accessed: 2024-03-13. NVM Express. https://nvmexpress.org

[11] Accessed: 2024-03-13. Samsung 980 PRO PCIe 4.0 SSD. https://semiconductor.samsung.com/consumer-storage/internal-ssd/980pro/

[12] Accessed: 2024-03-13. Toshiba Memory Introduces XL-FLASH Storage Class Memory Solution. https://americas.kioxia.com/en-us/business/news/2019/memory-20190805-1.html

[13] Accessed: 2024-03-13. Two New Block I/O Schedulers for 4.12. https://lwn.net/Articles/720675/

[14] Jens Axboe. Accessed: 2024-03-13. Efficient I/O with io_uring. https://kernel.dk/io_uring.pdf

[15] Alan J. Demers, Srinivasan Keshav, and Scott Shenker. 1989. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of the ACM Symposium on Communications Architectures & Protocols, SIGCOMM 1989*. ACM, 1–12.

[16] Diego Didona, Nikolas Ioannou, Radu Stoica, and Kornilios Kourtis. 2020. Toward a Better Understanding and Evaluation of Tree Structures on Flash SSDs. *Proc. VLDB Endow.* 14, 3 (2020), 364–377.

[17] Nima Elyasi, Mohammad Arjomand, Anand Sivasubramaniam, Mahmut T. Kandemir, Chita R. Das, and Myoungsoo Jung. 2017. Exploiting Intra-Request Slack to Improve SSD Performance. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017*. ACM, 375–388.

[18] Pawan Goyal, Harrick M. Vin, and Haichen Cheng. 1996. Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of the ACM SIGCOMM 1996 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 1996*. ACM, 157–168.

[19] Jiayang Guo, Yimin Hu, and Bo Mao. 2015. Enhancing I/O Scheduler Performance by Exploiting Internal Parallelism of SSDs. In *Algorithms and Architectures for Parallel Processing - 15th International Conference, ICA3PP 2015. Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 9531)*. Springer, 118–130.

[20] Jiayang Guo, Yiming Hu, Bo Mao, and Suzhen Wu. 2017. Parallelism and Garbage Collection Aware I/O Scheduler with Improved SSD Performance. In *2017 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2017*. IEEE Computer Society, 1184–1193.

[21] Mohammad Hedayati, Kai Shen, Michael L. Scott, and Mike Marty. 2019. Multi-Queue Fair Queuing. In *2019 USENIX Annual Technical Conference, USENIX ATC 2019*. USENIX Association, 301–314.

[22] Christoph Hellwig. 2009. XFS: The Big Storage File System for Linux. *login Usenix Mag.* 34, 5 (2009).

[23] Tejun Heo, Dan Schatzberg, Andrew Newell, Song Liu, Saravanan Dhakshinamurthy, Iyswarya Narayanan, Josef Bacik, Chris Mason, Chunqiang Tang, and Dimitrios Skarlatos. 2022. IOCost: Block IO Control for Containers in Datacenters. In *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating System 2022*. ACM, 595–608.

[24] Jaehyun Hwang, Midhul Vuppalapati, Simon Peter, and Rachit Agarwal. 2021. Rearchitecting Linux Storage Stack for $\mu$s Latency and High Throughput. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021*. USENIX Association, 113–128.

[25] Kanchan Joshi, Kaushal Yadav, and Praval Choudhary. 2017. Enabling NVMe WRR Support in Linux Block Layer. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*. USENIX Association.

[26] Myoungsoo Jung, Wonil Choi, Shekhar Srikantaiah, Joonhyuk Yoo, and Mahmut T. Kandemir. 2014. HIOS: A Host Interface I/O Scheduler for Solid State Disks. In *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014*. IEEE Computer Society, 289–300.

[27] Jieun Kim, Dohyun Kim, and Youjip Won. 2022. Fair I/O Scheduler for Alleviating Read/Write Interference by Forced Unit Access in Flash Memory. In *HotStorage '22: 14th ACM Workshop on Hot Topics in Storage and File Systems, 2022*. ACM, 86–92.

[28] Changman Lee, Dongho Sim, Joo Young Hwang, and Sangyeun Cho. 2015. F2FS: A New File System for Flash Storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST 2015*. USENIX Association, 273–286.

[29] Shaohong Li, Xi Wang, Xiao Zhang, Vasileios Kontorinis, Sreekumar Kodakara, David Lo, and Parthasarathy Ranganathan. 2020. Thunderbolt: Throughput-Optimized, Quality-of-Service-Aware Power Capping at Scale. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020*. USENIX Association, 1241–1255.

[30] Heiner Litz, Javier Gonzalez, Ana Klimovic, and Christos Kozyrakis. 2022. RAIL: Predictable, Low Tail Latency for NVMe Flash. *ACM Trans. Storage* 18, 1 (2022), 5:1–5:21.

[31] Mingzhe Liu, Haikun Liu, Chencheng Ye, Xiaofei Liao, Hai Jin, Yu Zhang, Ran Zheng, and Liting Hu. 2022. Towards Low-Latency I/O Services for Mixed Workloads Using Ultra-Low Latency SSDs. In *ICS '22: 2022 International Conference on Supercomputing, 2022*. ACM, 13:1–13:12.

[32] Hui Lu, Brendan Saltaformaggio, Ramana Rao Kompella, and Dongyan Xu. 2015. vFair: Latency-Aware Fair Storage Scheduling via per-IO Cost-Based Differentiation. In *Proceedings of the Sixth ACM Symposium on Cloud Computing, SoCC 2015*. ACM, 125–138.

[33] Till Miemietz, Hannes Weisbach, Michael Roitzsch, and Hermann Härtig. 2019. K2: Work-Constraining Scheduling of NVMe-Attached Storage. In *IEEE Real-Time Systems Symposium, RTSS 2019*. IEEE, 56–68.

[34] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving High CPU Efficiency for Latency-Sensitive Datacenter Workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019*. USENIX Association, 361–378.

[35] Stan Park and Kai Shen. 2012. FIOS: A Fair, Efficient Flash I/O Scheduler. In *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012*. USENIX Association, 13.

[36] Bo Peng, Haozhong Zhang, Jianguo Yao, Yaozu Dong, Yu Xu, and Haibing Guan. 2018. MDev-NVMe: A NVMe Storage Virtualization Solution with Mediated Pass-Through. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018*. USENIX Association, 665–676.

[37] Zebin Ren, Krijn Doekemeijer, Nick Tehrany, and Animesh Trivedi. 2024. BFQ, Multiqueue-Deadline, or Kyber? Performance Characterization of Linux Storage Schedulers in the NVMe Era. *To Appear in the Proceedings of the 2024 ACM/SPEC International Conference on Performance Engineering, ICPE 2024*. (2024).

[38] Zebin Ren and Animesh Trivedi. 2023. Performance Characterization of Modern Storage Stacks: POSIX I/O, libaio, SPDK, and io_uring. In *Proceedings of the 3rd Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems, CHEOPS 2023*. ACM, 35–45.

[39] Kai Shen and Stan Park. 2013. FlashFQ: A Fair Queueing I/O Scheduler for Flash-Based SSDs. In *2013 USENIX Annual Technical Conference, 2013*. USENIX Association, 67–78.

[40] Matthew Wachs, Michael Abd-El-Malek, Eno Thereska, and Gregory R. Ganger. 2007. Argon: Performance Insulation for Shared Storage Servers. In *5th USENIX Conference on File and Storage Technologies, FAST 2007*. USENIX, 61–76.

[41] Mingyang Wang and Yiming Hu. 2014. An I/O Scheduler Based on Fine-Grained Access Patterns to Improve SSD Performance and Lifespan. In *Symposium on Applied Computing, SAC 2014*. ACM, 1511–1516.

[42] Caeden Whitaker, Sidharth Sundar, Bryan Harris, and Nihat Altiparmak. 2023. Do We Still Need IO Schedulers for Low-latency Disks?. In *Proceedings of the 15th ACM/USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2023*. ACM, 44–50.

[43] Jiwon Woo, Minwoo Ahn, Gyusun Lee, and Jinkyu Jeong. 2021. D2FQ: Device-Direct Fair Queueing for NVMe SSDs. In *19th USENIX Conference on File and Storage Technologies, FAST 2021*. USENIX Association, 403–415.

[44] Kan Wu, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2019. Towards an Unwritten Contract of Intel Optane SSD. In *11th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2019*. USENIX Association.

[45] Suzhen Wu, Weiwei Zhang, Bo Mao, and Hong Jiang. 2019. HotR: Alleviating Read/Write Interference with Hot Read Data Replication for Flash Storage. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019*. IEEE, 1367–1372.

[46] Qiumin Xu, Huzefa Siyamwala, Mrinmoy Ghosh, Tameesh Suri, Manu Awasthi, Zvika Guz, Anahita Shayesteh, and Vijay Balakrishnan. 2015. Performance Analysis of NVMe SSDs and Their Implication on Real World Databases. In *Proceedings of the 8th ACM International Systems and Storage Conference, SYSTOR 2015*. ACM, 6:1–6:11.

[47] Jinfeng Yang, Bingzhe Li, and David J. Lilja. 2020. Exploring Performance Characteristics of the Optane 3D Xpoint Storage Technology. *ACM Trans. Model. Perform. Evaluation Comput. Syst.* 5, 1 (2020), 4:1–4:28.

[48] Minhoon Yi, Minho Lee, and Young Ik Eom. 2017. CFFQ: I/O Scheduler for Providing Fairness and High Performance in SSD Devices. In *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, IMCOM 2017*. ACM, 87.

# Baking Disaster-Proof Kubernetes Applications with Efficient Recipes

Runyu Jin
runyu.jin@ibm.com
IBM Almaden Research Center
Almaden, California, USA

Paul Muench
pmuench@us.ibm.com
IBM Almaden Research Center
Almaden, California, USA

Travis Janssen
travis.janssen@ibm.com
IBM Almaden Research Center
Almaden, California, USA

Brian Hatfield
bhatfiel@us.ibm.com
IBM Almaden Research Center
Almaden, California, USA

Veera Deenadhayalan
veerad@us.ibm.com
IBM Almaden Research Center
Almaden, California, USA

## ABSTRACT

Multicluster disaster recovery on cloud-native platforms such as Kubernetes usually replicates application data and Kubernetes resources to a safe recovery cluster. In the event of a disaster, Kubernetes resources are restored to the recovery cluster to recover the affected applications. We tested 10 popular Kubernetes applications using this naive approach, and 60% failed. Problems include data being restored in the wrong order, cluster-specific data being restored instead of generated by the cluster, etc. All these problems lead to our recipe design that enables disaster recovery of all Kubernetes applications. In this paper, we analyze the problems we encountered during the disaster recovery of Kubernetes applications and categorize applications based on their disaster recovery behaviors. We present a recipe that groups, orders, and filters Kubernetes resources to enable disaster recovery. Finally, we evaluate the reliability and efficiency of the recipe. Our evaluation shows that recipe achieves a 100% success rate of disaster recovery while adding mere seconds of overhead to the recovery time.

## CCS CONCEPTS

• **Computer systems organization** → **Reliability**; **Cloud computing**; **Availability**.

## KEYWORDS

Disaster Recovery, Cloud Computing, Kubernetes, Reliability, Containerized, Multi-cluster

**ACM Reference Format:**
Runyu Jin, Paul Muench, Travis Janssen, Brian Hatfield, and Veera Deenadhayalan. 2024. Baking Disaster-Proof Kubernetes Applications with Efficient Recipes. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651417

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651417

## 1 INTRODUCTION

Kubernetes has become the container orchestration platform of choice across the IT industry [26]. Kubernetes is no longer focused only on stateless applications and as a result the users of Kubernetes are concerned with data resiliency [25]. This paper focuses on disaster recovery (DR). A naive approach to protecting stateful applications against disasters is to replicate all application persistent storage and all Kubernetes resources to a safe recovery cluster. This naive approach failed for 60% of 10 stateful applications evaluated. This paper proposes disaster recovery recipes, which is a method of implementing disaster recovery for Kubernetes applications for which the naive approach fails.

Recipes in our disaster recovery solution have 4 goals: (1) provide a disaster recovery solution to currently deployed applications (2) enable Site Reliability Engineers or developers to create disaster recovery recipes for applications (3) enable application developers to create disaster recovery hooks when other recovery techniques are not sufficient (4) make application disaster recovery reliable and efficient. Achieving all of these goals requires a user to deploy a disaster recovery framework along with Kubernetes as Kubernetes is not inherently disaster resistant. The disaster recovery framework discussed in this paper is Ramen [20], which is an IBM open-source project that provides a naive DR solution to Kubernetes applications. We implemented recipes in Ramen and evaluated the recipe design using Ramen. The resulting implementation is now available on the Ramen GitHub.

Ten Kubernetes data management system applications were studied. Four of the data management systems were able to recover from 100% of simulated disasters without a recipe. Two of the data management systems required recipes without hooks to achieve any successful recovery from a disaster. The remaining four applications require both recipes and hooks and that recovery technique is not the focus of this paper. With 1 to 3 days effort each of the 2 applications that needed recipes without hooks recovered from 100% of simulated disasters. On average the Kubernetes resource restore time for applications is 28% of the application's recovery time. Our DR solution has a small and fixed recovery time for any size of raw data. Thus our approach is both highly effective and highly efficient.

Our disaster recovery solution with recipes is the only known solution that can combine filtering, ordering, and hooks for Kubernetes resource protection and recovery. These protection and

recovery mechanisms enable two essential values. First, our solution can be applied to running applications. Second, our solution can be leveraged by users without changing application code which can make a solution rapidly available. Other solutions exist to protect Kubernetes applications from disasters. There are backup/restore solutions that require time to restore data before an application is restarted [21], which makes total recovery time longer. There are solutions based on cross data center Kubernetes clusters [19], but those solutions rely on synchronous replication. Synchronous replication is not suitable to address region wide disasters as the communication latency for maintaining cluster membership across regions is too high. There are other solutions like ours that can protect applications across regions with data in place recovery [30]. However, these other solutions do not provide the flexible Kubernetes resource protection and recovery mechanisms that are needed by data management systems.

## 2 BACKGROUND

Business continuity is the ability of a business to meet the demands of its clients by recovering from diverse types of problems as quickly as possible at a reasonable cost. The types of problems that affect business continuity can be broadly classified as follows: (1) failure of subsystems (such as, nodes, network, etc.) within a data center or availability zone (AZ), (2) unrecoverable corruption of data (either accidentally or maliciously), and (3) failure of the entire data center or multiple data centers in a region. These problems are solved respectively using: (a) high availability (HA) solutions, such as redundancy within the data center to avoid single points of failure, (b) backup and restore (B/R) of data using an external secondary store, and (c) disaster recovery (DR) solutions that replicate the application state to another data center that is in a different AZ or region [24]. In our paper, we focus on DR but some of the problems we outline in Section 3 also apply to B/R.

The goal of B/R and DR solutions is to recover critical application state after a disaster event. Bare-metal servers, virtual machines, and containers typically store their bootstrap configuration data in locally attached persistent volumes (e.g., in the `/etc` directory of the root file system) and may store their application volume data either in locally attached persistent volumes or in remote network attached persistent volumes. Whereas, in Kubernetes, application state is a combination of (a) Kubernetes API resources stored in an `etcd` backing store [17] and (b) volume data stored in persistent volumes. Examples of API resource types are deployments, pods, PersistentVolumeClaims (PVCs), PersistentVolumes (PVs), services, secrets, and ConfigMaps.

There are many DR solutions for Kubernetes. Kubernetes stretched cluster is a solution that takes a single Kubernetes cluster and stretches it by placing all the control plane components, including etcd, across AZs within a region. This solution leverages the HA features of Kubernetes to build a basic DR solution across multiple AZs [18]. Given that this solution uses a single etcd cluster across multiple AZs, it does not suffer from the multi-cluster DR problems we motivate in Section 3. However, this solution suffers from the effects of network latency across AZs. The longer the distance between the AZs, the higher the latency is between them and proportionally severe is the latency effect on etcd replication I/O across AZs. High network latencies can cause etcd to miss heartbeats, experience timeouts, result in leader elections that are disruptive to the cluster, and can also lead to API slowness [35]. This makes it unsuitable to stretch the cluster across long distances. Hence, this limitation of Kubernetes stretched cluster serves as a motivation for Kubernetes multi-cluster DR, which is the focus of this paper. Kubernetes multi-cluster DR [34] is a solution that overcomes the limitations of Kubernetes stretched cluster by using multiple independent Kubernetes clusters, each with its own etcd backing store.

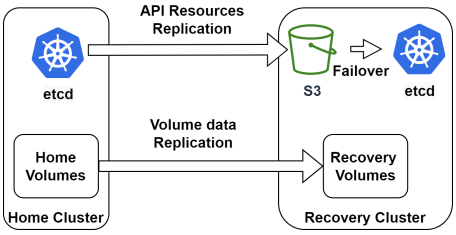



**Figure 1: Kubernetes Multi-cluster DR**

Kubernetes enables building DR features with its extensible design paradigm. While DR features are not part of core Kubernetes, many vendors offer custom extensions to Kubernetes. Ramen [20] is an example of an open-source DR solution. The Ramen DR solution can handle applications deployed using GitOps [7] and other traditional deployment methods. It focuses on protection and recovery of Kubernetes API resources only. To protect API resources, Ramen captures selected API resources, asynchronously stores them in an object store in the recovery DR cluster, and restores them to the etcd of the recovery DR cluster after a disaster event.

## 3 NAIVE KUBERNETES DR

Our primary motivation is to disaster-proof stateful Kubernetes applications (a) without modifying the applications themselves and (b) irrespective of the application deployment methodology [2].

### 3.1 Limitations Of The Naive Approach

GitOps [7] uses CI/CD pipelines to automate deployment of applications using declarative object-configuration stored in Git repositories. A naive approach to recover such a GitOps deployed application is to again use GitOps on the recovery cluster. However, this approach may not work for all types of applications. Consider an example of a GitOps deployed stateful application that has PVCs. Kubernetes will dynamically provision PV resources to fulfill the PVCs. These locally created PV resources in the home cluster do not come from the declarative Git source. If one were to use GitOps in the recovery cluster, Kubernetes will dynamically provision new PV resources to fulfill the PVCs that do not contain PV contents at the home cluster, resulting in data loss. We expect other application specific examples where not all API resources come from a declarative source, which makes GitOps-based DR fall short as a full DR solution. This implies that we need to capture the API resources stored in the home cluster and restore it to the recovery cluster.

A naive capture of the API resources on the home cluster periodically queries the API server and stores the results in the recovery cluster. After a disaster event, a naive recovery of API resources

on the recovery cluster restores previously captured API resources of the home cluster to the recovery cluster. This naive approach does not consider parent and child resources, sequence resource creation, nor exclude any resources.

We define a naive DR approach to use naive capture, naive recovery, or use naive GitOps for DR. While naive DR appears to be simple and intuitive, our case-study shows examples where naive DR falls short due to problems that come from (a) the application (b) the third-party DR service that is in use, (c) Kubernetes itself. Kubernetes aims to eliminate the need for orchestration of complex applications with its design comprising a set of independent, composable control processes that continuously drive the current state towards the provided desired state, but these principles do not apply to diverse applications that are out in the wild.

## 3.2 Case Study Using The Naive Approach

We selected 10 popular Kubernetes applications to study how the naive approach works. Five are database management systems ranked among DB-Engine's seven most popular either directly or as open-source relatives: Elasticsearch [8], EnterpriseDB [12], MariaDB [14], MongoDB [27], and Redis [36, 39]. Two are popular open-source machine learning frameworks: PyTorch [31] and TensorFlow [1, 40]. Jenkins [22] is one of the most popular continuous integration/continuous delivery (CI/CD) tools [6]. Apache Kafka [13] is the most popular open-source event streaming platform [23]. And Apache Spark [15] is an engine for large-scale data processing used by 80% of Fortune 500 companies [15].

The naive approach failed to recover 60% of the applications from simulated disaster. We categorize the reasons for failure into four modes. Table 1 below lists the applications tested and for which reasons they failed.

| | | Naive Approach Failure Reason | | | |
|---|---|---|---|---|---|
| Application | Success | Absence | Order | Stale | Mode |
| Elasticsearch | | ✓ | ✓ | | |
| EnterpriseDB | | | | | ✓ |
| Jenkins | ✓ | | | | |
| Kafka | ✓ | | | | |
| MariaDB | | | | ✓ | |
| MongoDB | ✓ | | | | |
| PyTorch | | | | ✓ | |
| Redis | | | | | ✓ |
| Spark | ✓ | | | | |
| TensorFlow | | | | ✓ | |

**Table 1: Naive Approach DR Results**

Following are examples of how applications failed to recover from simulated disaster, including some analysis of why the failures happened. Each example failure is classified according to the failure reason in Table 1. An application that fails to recover due to *absence* requires a Kubernetes API resource that is missing. For example, Elasticsearch could not query `ApmServer` resources because its custom resource definition (CRD) resource was not installed. The CRD resource type is cluster-scoped and the naive approach restores cluster-scoped resources by exception only. A CRD is only restored if a resource of the type it defines exists in the application's namespace [5].

The naive approach restores most resource types in alphabetical order by type name [4], e.g., `Deployments`, `Jobs`, `StatefulSets`. It recreates resources without delay between types, but some applications benefit from a different order or delay. The *order* column identifies applications that failed to recover due to their Kubernetes API resources being restored in an incorrect order. For example, Elasticsearch typically failed to reach healthy status when an `EnterpriseSearch` resource was restored immediately after its associated `Elasticsearch` one, but succeeded whenever they were separated by a five-second delay [9]. We discovered this accidentally by specifying an explicit restore order which introduced the delay.

Some applications failed to recover when *stale* information specific to the home cluster persisted in their Kubernetes API resources. For example, PyTorch, deployed by the OpenShift Data Science operator, failed to recover because an endpoint, containing the home cluster's name, was unreachable. The application recovered after replacing the home cluster's name with the recovery cluster's. Some applications require a specific *mode* to restart on another cluster. Redis, for example, failed to restart with the naive approach, but succeeded when its `RedisEnterpriseCluster` resource was modified setting its `spec.clusterRecovery` mode to `true` [37]. In summary, the naive approach to DR worked for some applications, but not all. The naive approach worked for applications that omitted cluster-specific information, did not have restore order requirements, had no dependencies, and generally tolerated being restored on another cluster.

# 4 ROBUST KUBERNETES DR USING RECIPES

Theoretically, Kubernetes application resources can be backed up as a single unordered group, then restored as a single unordered group, and eventually regain a functional state, provided the application data is available. Our case study indicates that this assumption does not hold true for many applications. Further, the issues that prevent backup or recovery in a single group may not be fixed without modifying the underlying application. To address these issues, we introduce the Recipe concept for robust disaster recovery.

## 4.1 Recipe

A recipe is a Kubernetes custom resource (CR) that defines the capture and recovery sequences of Kubernetes objects. Recipes enable and automate DR for any application. The recipe design specifically addresses the issues experienced with naive DR as discussed in Section 3. A key abstraction in a recipe is a workflow, which defines a sequence of actions to take during a capture or recovery sequence. A workflow is a sequentially processed list of groups and hooks. Groups define the resources to be included or excluded in a step of a workflow, and Hooks define actions that should be run in between groups. With these three abstractions, all issues encountered during the case study can be addressed.

The absence issue encountered with ElasticSearch involved a missing CRD (ApmServer resource) whose absence failed the recovery. Backing up an object CR requires the CRD, as does restoring it. A recipe can handle this situation by capturing an active object (ApmServer CR) along with the CRD it uses. This is preferred to

installing the full operator which includes the CRD on the recovery cluster, as the recovery cluster version may not match the operator version without prior planning.

Combinations of groups enable sequencing a recovery process. If a particular resource is dependent on another in a parent-child relationship, they can be split up into two groups to ensure the parent resource exists before the child resource. Groups address the absence and order problems encountered with ElasticSearch, where the ElasticSearchCluster CR needs to be available before the EnterpriseSearch CR (Recipe Example 1).

Hooks can address scenarios where an object must be modified after restoration [3]. Restoring an object involves copying the resource contents from the home cluster and reproducing them on the recovery cluster. Some applications, like OpenShift Data Science, embed cluster-specific information in their resources, like Notebooks. Kubernetes resources can become available, but the application is inaccessible through an endpoint, which uses a stale URL. By correcting this data, Hooks can address the stale issues encountered in the case study.

Hooks can also address scenarios where selectively ordering or filtering resource groups is insufficient to recover an application. Redis requires that a recovery mode is specified on the CR when a majority of nodes become unavailable [37]. During recovery, Redis begins without any nodes available, and setting the recovery mode is required to launch pods and continue operation. Adding spec.clusterRecovery=true to the RedisEnterpriseCluster CR to begin the recovery mode that is required by Redis. Hooks can be used to add the recovery mode field on application CRs, addressing mode-type scenarios found during the case study.

## 4.2 Recipe API and Examples

Now that the high level abstractions have been explained, the API can be introduced. The sample recipe is based on ElasticSearch, but adds hooks to demonstrate the feature. The recipe object itself is divided into the three abstractions: groups, hooks and workflows. The captureWorkflow is used for the backup/capture sequence, and the recoverWorkflow is used for the restore/recover sequence. A sequence is defined using a map of strings, where a user specifies a type (Group or Hook), then the name of that group or hook (for example: "group: everything"). Each step of the sequence must be completed before the next one begins.

Groups are defined with a unique name identifier, and may include and exclude resource types by name. Groups use namespace-scoped visibility by default, but may opt-into cluster-scoped resources with an additional field (includeClusterResources = true). Since capture and recovery sequences may not be symmetrical, a backupRef field is used to source recovery contents.

A full Recipe is shown below in Recipe Example 1. In the current recipe implementation, groups are processed independently. Therefore, it is possible to restore duplicate resources across different groups. Using excludedResourceTypes avoids this scenario.

**Recipe Example 1**

```
apiVersion: ramendr.openshift.io/v1alpha1
kind: Recipe
metadata:
  name: recipe-demo
  namespace: eck
spec:
  appType: eck
  volumes:
    name: volumes
    type: volume
    labelSelector: {} # select all PVCs
  groups:
  - name: everything
    type: resource
    includedResourceTypes:
    - "*"
  - name: cluster
    backupRef: everything
    type: resource
    includedResourceTypes:
    - elasticsearches.elasticsearch.k8s.elastic.co
  - name: enterprise-search
    backupRef: everything
    type: resource
    includedResourceTypes:
    - enterprisesearches.enterprisesearch\
      .k8s.elastic.co
  - name: misc
    backupRef: everything
    type: resource
    excludedResourceTypes:
    - enterprisesearches.enterprisesearch\
      .k8s.elastic.co
    - elasticsearches.elasticsearch.k8s.elastic.co
  hooks:
  - name: demo-hooks
    labelSelector:
      matchLabels:
        appname: eck
    type: exec
    ops:
    - name: date
      container: main
      timeout: 10m
    command: # runs as single command: "/bin/sh -c date"
      - "/bin/sh"
      - "-c"
      - "date"
  captureWorkflow:
    sequence:
    - hook: demo-hooks/date
    - group: everything
  recoverWorkflow:
    sequence:
    - group: cluster
    - group: enterprise-search
    - group: misc
```

## 4.3 Implementation

Recipes are a general concept which may be used as a library component [10]. The controller logic discussed in this paper was implemented within Ramen, an open-source Disaster Recovery solution

| Category | Application |
|---|---|
| Type 1 | Jenkins, Kafka, MongoDB, Spark |
| Type 2 | Elasticsearch, MariaDB |
| Type 3 | EnterpriseDB, PyTorch, Redis, Tensorflow |

**Table 2: Kubernetes Applications Categorization**

[20]. Ramen handles volumes replication across clusters, while recipes handle application recovery. The separation of API and controller logic allows for customization of the software stack, if a user desires. Ramen was released as a part of OpenShift Data Foundations (formerly OpenShift Container Storage) v4.7 [32].

## 5 EVALUATION

We evaluated the performance of recipe-based disaster recovery following the design described in Section 4. We implemented the recipe based on Ramen [20]. Our key evaluation metrics are reliability and efficiency of application recovery.

### 5.1 Environment Setup

We setup two RedHat Openshift Container Platform (OCP) [16] 4.12 clusters, one cluster serves as the home cluster where applications are installed and initially deployed. Another cluster serves as the recovery cluster. When a simulated disaster happens, all the applications deployed on the home cluster are recovered on the recovery cluster. The application deployed on the home cluster is removed completely including all the Kubernetes resources and persistent volumes to mimic real-world production outages. Both clusters use the same external RedHat OpenShift Data Foundation (ODF) Ceph [33] cluster as the storage backend for all the application persistent data. ODF offers the Metro Disaster Recovery (MetroDR) solution which synchronously replicates persistent data between the home and recovery clusters. We deployed Ramen on both OCP clusters to protect Kubernetes application resources every 5 minutes. We simulate a disaster scenario by removing all the applications on the home cluster, and then initiate disaster recovery on the recovery cluster.

### 5.2 Kubernetes Applications Categorization

Applications are categorized into three types, which reflects the amount of custom handling required with a Recipe to enable DR protection for the application. Type 1 applications do not require Recipes: all the resources can be captured and recovered on its own. Type 2 applications require Recipe Groups: either resource filtering and/or ordering are required to restore the application. Type 3 applications require both Recipe Groups and Hooks: if an application requires a program to run to reach a consistent application state, this is done with a Hook. The categorization of the 10 applications is shown in Table 2 based on the naive DR approach we studied (Section 3).

We focus the evaluation on type 1 and type 2 applications in this paper and type 3 applications evaluation will be future work. We evaluated 6 modern applications that are either type 1 or type 2 applications which are a subset of the 10 applications introduced in Section 3. The applications include Elasticsearch (v2.8.0), Jenkins (v2.432), Apache Kafka (v2.5.0), MariaDB (v0.20.0), MongoDB (v7.0), and Apache Spark (v3.1.1). Among them, Elasticsearch and MariaDB are type 2 applications and use recipes. Elasticsearch uses 3 recipe groups for ordering while MariaDB uses 4 recipe groups for both ordering and filtering. All other applications are type 1 applications and do not use recipes.

| Application | Ramen w/o Recipes | Ramen w Recipes |
|---|---|---|
| Elasticsearch | 0% | 100% |
| Jenkins | 100% | 100% |
| Kafka | 100% | 100% |
| MariaDB | 0% | 100% |
| MongoDB | 100% | 100% |
| Spark | 100% | 100% |

**Table 3: DR Success Rate Without & With A Recipe**

During the evaluation, each application is tested alone for its disaster recovery reliability and efficiency. We tested disaster recovery 40 times one way for each application from home cluster to recovery cluster and then back from recovery cluster to home cluster back to back. Each disaster recovery iteration consists of Kubernetes resource restore time and application recovery time. Kubernetes resource restore time is the time for Ramen to create all the Kubernetes resources. Ramen's role in application recovery ends once the Kubernetes objects have been created on the recovery cluster. Application recovery time includes the Kubernetes resource restore time, but also includes the time for the application to become fully usable. We will show the success rate and the recovery efficiency for the 80 runs in the following sections.

### 5.3 Recipe Reliability And Efficiency

Table 3 shows the success rate for the 40 runs of disaster recovery of the 6 applications. Without recipe, Elasticsearch and MariaDB cannot be successfully recovered. All the type 1 applications were able to achieve 100% success rate of recovery. After applying the recipe, Elasticsearch and MariaDB recovered successfully 100% of the time. Also, the code modifications to Ramen to include the new recipe design do not reduce the reliability of type 1 applications, all the applications were recovered 100% of the time.

Figure 2 shows the the total recovery time v.s. the Kubernetes resource restore time for the six applications. All the applications were able to complete disaster recovery within 5 minutes. The longest one was Kafka, which takes around 265 seconds. The shortest one was MongoDB, which takes around 106 seconds. Kubernetes resource restore time takes a small portion of the total recovery time, with an average of 28%. For most applications, it takes around 20% of the total application recovery time to restore Kubernetes resources. MongoDB and Jenkins finish Kubernetes resources restore the fastest with 29 seconds. MariaDB takes the longest to restore Kubernetes resources with 92 seconds around 61% of the total recovery time. All the applications were able to finish recovering Kubernetes resources within one and half minutes. Note that besides the resource size, resource types also make a difference to the resource restore time. Certain resources like pods take longer to recover due to the underlying design of Velero where the restore process requires querying of the server object by object.

After Ramen restores all the resources, it takes some more time for the application to become ready by reconciling the application

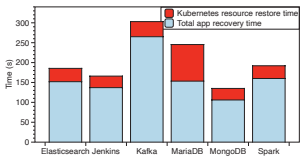



**Figure 2: App DR Time v.s. Resource Restore Time**

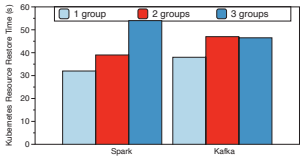



**Figure 3: Adding Recipe Groups Overhead**

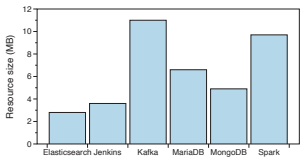



**Figure 4: Application Resource Size In Megabytes**

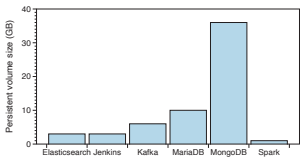



**Figure 5: Application Data Size In Gigabytes**

to match the desired application state in the Kubernetes resources. MariaDB takes the shortest time around 58 seconds to become ready. Kafka takes the longest of 227 seconds to become ready. Compared to other applications, Kafka has two sets of pods, zookeeper and kafka, to reconcile. MariaDB and Elasticsearch which are type 2 applications do not have a substantially longer recovery time compared to other type 1 applications.

To measure the overhead of using recipe groups, we forcefully added recipe groups to type 1 applications and show the Kubernetes resource restore time after adding different number of recipe groups. Figure 3 illustrates the results. Adding 1 group has the same Kubernetes resource restore time as not using recipe. When we add more recipe groups, the Kubernetes resource restore time increases. For Spark, from 1 group to 2 groups increases the restore time by 7 seconds. It further increases the restore time by 15 seconds from 2 groups to 3 groups. For Kafka, 2 groups and 3 groups add around 9 seconds of overhead. Most of the overhead come from communicating with API server or Kubelet when switching groups which takes around 5 seconds. We haven't tuned API server to be efficient and we believe this overhead can be eliminated after optimization. Given the reliability that recipe enhanced, recipe overhead is low in seconds and still restores resources with a reasonable amount of time.

The total recovery time is largely related with the amount of Kubernetes resources each application has. Figure 4 shows the size of the Kubernetes resources in Megabytes. In general, the larger the resource size, the longer it takes to recover the application. We can see Kafka has the largest amount of resources so it takes the longest to recover. For Elasticsearch and MariaDB which are type 2 applications, recipe adds some overhead to the total recovery time.

Figure 5 shows the total data size of all the Persistent Volumes (PV) for each application. Because we are using MetroDR which synchronously backups data volumes across the clusters, the data volumes don't need to be restored during disaster recovery. This brings the benefits that the disaster recovery time is not related to the data size of the PVs. Although MongoDB has the largest PV size, it doesn't take the longest to restore MongoDB.

## 6 RELATED WORKS

Disaster recovery for Kubernetes applications is new enough that there are no standards for how it should be deployed. Disaster recovery solutions can be divided into two classes, solutions for stateless applications and solutions for stateful applications. At this time the techniques used for each class of disaster recovery solution are distinct, but how techniques can be reused between the two classes is an open area of research. So this section will compare our solution with both classes of application disaster recovery.

The most common disaster recovery approach for Kubernetes stateful applications is to use a backup/restore solution with persistent volumes and Kubernetes resources being protected on a remote site. De et al [11], Pakrijauskas and Mažeika [29] and Rubio [38] all evaluate backup/restore solutions in the cloud. These studies do not focus on evaluating successful disaster recovery for a set of Kubernetes applications. This can explain why De et al [11], Pakrijauskas and Mažeika [29] and Rubio [38] do not mention the need for a disaster recovery solution based on more than simple replication. Torta [41] focuses on disaster recovery managed by data management systems. Due to the active-active nature of data management system disaster recovery the associated recovery time can be very small. The limitation of using the disaster recovery strategy in Torta [41] is that every application must have its own disaster recovery solution. This application by application approach to disaster recovery can be complex to manage and hence risky to operate. Tran et al [42] investigates the mechanisms for the protection and recovery of running containers based based on application checkpoints. Tran et al [42] does not investigate the protection and recovery of Kubernetes applications with persistent volumes and Kubernetes resources.

Stateless applications do not use persistent volumes and do not rely on updates to Kubernetes resources. Hence these applications can easily be recovered after a disaster. Moshfeghifar [28] studies stateless applications in the form of serverless computing. The key challenges addressed in Moshfeghifar [28] are deploying applications across multiple clusters and getting those clusters to act like a single cluster environment.

## 7 CONCLUSION AND FUTURE WORK

This paper presents a disaster recovery (DR) solution for Kubernetes. The novelties of this work lie in first, categorization of the problems that current modern Kubernetes applications have when doing DR; second, present a novel disaster recovery solution called recipes to enable DR for all the modern applications without modifications to the applications; third, evaluate the recipe solution to confirm that recipe can achieve 100% success rate of DR with low overhead. In our future work, we will further explore recipe design with hooks and how hooks can help disaster recovery of type 3 applications. Besides, we will explore the disaster recovery solution's applicability for large scale database deployments and databases under continuous load. We will also validate application data staleness.

## REFERENCES

[1] Omdia Analyst. 2022. *The Evolution of ML Frameworks Report - 2022.* Technical Report. Omdia.
[2] The Kubernetes Authors. 2023. Kubernetes Object Management. https://kubernetes.io/docs/concepts/overview/working-with-objects/object-

management/
[3] Velero Authors. 2023. Velero 1.12 Restore Resource Modifiers. https://velero.io/docs/v1.12/restore-resource-modifiers/
[4] Velero Authors. 2023. Velero Docs - Restore Reference. https://velero.io/docs/v1.9/restore-reference/#restore-order
[5] Velero Authors. 2023. velero/pkg/backup/backup.go at 9b5678f32a4aa696de5d645d15bc0ff1f989f464 · vmware-tanzu/velero. https://github.com/vmware-tanzu/velero/blob/9b5678f32a4aa696de5d645d15bc0ff1f989f464/pkg/backup/backup.go#L410-L419
[6] Michael Azoff. 2023. *Omdia Universe: DevOps Release Management Solutions, 2023.* Technical Report. Omdia.
[7] Florian Beetz and Simon Harrer. 2022. GitOps: The Evolution of DevOps? *IEEE Software* 39, 4 (2022), 70–75. https://doi.org/10.1109/MS.2021.3119106
[8] Elasticsearch B.V. 2022. Elasticsearch Platform — Find real-time answers at scale | Elastic. https://www.elastic.co/
[9] Elastic B.V. 2023. Prerequisites | Enterprise Search documentation [8.11] | Elastic. https://www.elastic.co/guide/en/enterprise-search/current/prerequisites.html#prerequisites
[10] IBM Corp. 2023. Recipe API. https://github.com/RamenDR/recipe/blob/main/api/v1alpha1/recipe_types.go
[11] Suman De, R Prashant Singh, et al. 2022. Selective Analogy of Mechanisms and Tools in Kubernetes Lifecycle for Disaster Recovery. In *2022 IEEE 2nd International Conference on Mobile Networks and Wireless Communications (ICMNWC)*. IEEE, IEEE, 3 Park Avenue, 17th Floor New York, NY 10016-5997 USA, 1–6.
[12] enterprisedb. 2023. EnterpriseDB. "https://www.enterprisedb.com/"
[13] Apache Software Foundation. 2022. Apache Kafka. https://kafka.apache.org/
[14] MariaDB Foundation. 2022. https://mariadb.org/. MariaDBServer: Theopensourcerelationaldatabase
[15] The Apache Software Foundation. 2022. Unified engine for large-scale data analytics. https://spark.apache.org/
[16] The Linux Foundation. 2022. OpenShift Container Platform 4.12 Documentation. https://docs.openshift.com/container-platform/4.12/welcome/index.html
[17] The Linux Foundation. 2023. Kubernetes Components. https://web.archive.org/web/20231025011453/https://kubernetes.io/docs/concepts/overview/components/#etcd
[18] The Linux Foundation. 2023. Kubernetes: Running in multiple zones. https://web.archive.org/web/20231020051135/https://kubernetes.io/docs/setup/best-practices/multiple-zones/
[19] Red Hat. 2023. OpenShift Disaster Recovery using Stretch Cluster. https://red-hat-storage.github.io/ocs-training/training/ocs4/ocs4-metro-stretched.html
[20] Red Hat. 2023. Ramen DR opensource project. https://github.com/RamenDR/ramen/
[21] IBM. 2021. Overview of Kubernetes Backup Support. https://www.ibm.com/docs/en/spp/10.1.5?topic=containers-overview
[22] Jenkins. 2022. Jenkins. https://www.jenkins.io/
[23] Alex Johnston. 2022. *Connectivity is the watchword as Confluent continues to expand.* Technical Report. 451 Research.
[24] Th. Lumpp, J. Schneider, J. Holtz, M. Mueller, N. Lenz, A. Biazetti, and D. Petersen. 2008. From high availability and disaster recovery to business continuity solutions. *IBM Systems Journal* 47, 4 (2008), 605–619. https://doi.org/10.1147/SJ.2008.5386516
[25] Parth Sandip Mehta. 2023. *NoSQL Databases in Kubernetes.* Master's thesis. San Jose State University. https://doi.org/10.31979/etd.qrrp-3equ
[26] Christine Miyachi. 2021. The Rise of Kubernetes. In *2021 Cloud Continuum*. IEEE, 3 Park Avenue, 17th Floor New York, NY 10016-5997 USA, 1–5. https://doi.org/10.1109/CloudContinuum54760.2021.00002
[27] Inc. MongoDB. 2022. MongoDB: For the next generation of intelligent applications. https://www.mongodb.com/
[28] Amirhossein Moshfeghifar. 2022. *Active Disaster Recovery Strategy for Applications Deployed Across Multiple Kubernetes Clusters, Using Service Mesh and Serverless Workloads.* Master's thesis. Tampere University.
[29] Kęstutis Pakrijauskas and Dalius Mažeika. 2021. On recent advances on stateful orchestrated container reliability. In *2021 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)*. IEEE, IEEE, 3 Park Avenue, 17th Floor New York, NY 10016-5997 USA, 1–6.
[30] Portworx. 2023. Disaster Recovery. https://docs.portworx.com/portworx-enterprise/operations/operate-kubernetes/disaster-recovery
[31] pytorch. 2023. PyTorch. "https://pytorch.org/"
[32] Inc. Red Hat. 2021. OpenShift Container Storage 4.7 release notes. https://access.redhat.com/documentation/en-us/red_hat_openshift_container_storage/4.7/html-single/4.7_release_notes/index
[33] Inc. Red Hat. 2022. Red Hat OpenShift Data Foundation. https://www.redhat.com/en/technologies/cloud-computing/openshift-data-foundation
[34] Inc. Red Hat. 2023. Introduction to OpenShift Data Foundation Disaster Recovery. https://web.archive.org/web/20231010175615/https://access.redhat.com/documentation/en-us/red_hat_openshift_data_foundation/4.13/html-single/configuring_openshift_data_foundation_disaster_recovery_for_openshift_workloads/index#introduction-to-odf-dr-solutions_common
[35] Inc Red Hat. 2024. Recommended etcd practices. https://web.archive.org/web/20231105012238/https://docs.openshift.com/container-platform/4.14/scalability_and_performance/recommended-performance-scale-practices/recommended-etcd-practices.html
[36] redis. 2023. https://redis.io/. Redis
[37] Redis. 2023. Recover a Redis Enterprise cluster on Kubernetes | Redis Documentation Center. https://docs.redis.com/latest/kubernetes/re-clusters/cluster-recovery/
[38] Sergio Fernández Rubio. 2022. *Disaster Recovery Analysis of different Cloud Managed Kubernetes Clusters.* Master's thesis. Edinburgh Napier University. https://www.researchgate.net/profile/Sergio-Fernandez-Rubio/publication/363632856_Disaster_Recovery_Analysis_of_different_Cloud_Managed_Kubernetes_Clusters/links/6325ee52873eca0c0094f0e1/Disaster-Recovery-Analysis-of-different-Cloud-Managed-Kubernetes-Clusters.pdf
[39] solid IT. 2023. DB-Engines Ranking. https://db-engines.com/en/ranking
[40] tensorflow. 2023. TensorFlow. "https://www.tensorflow.org/"
[41] Francesco Torta. 2023. *Business Continuity in Kubernetes Multi-Cluster Environments.* Ph. D. Dissertation. Politecnico di Torino.
[42] Minh-Ngoc Tran, Xuan Tuong Vu, and Younghan Kim. 2022. Proactive Stateful Fault-Tolerant System for Kubernetes Containerized Services. *IEEE Access* 10 (2022), 102181–102194.

# Empirical Evaluation of ML Models for Per-Job Power Prediction

Debajyoti Halder
Stony Brook University
Stony Brook, New York, USA
dhalder@cs.stonybrook.edu

Manas Acharya
Stony Brook University
Stony Brook, New York, USA
macharya@cs.stonybrook.edu

Aniket Malsane
Stony Brook University
Stony Brook, New York, USA
amalsane@cs.stonybrook.edu

Anshul Gandhi
Stony Brook University
Stony Brook, New York, USA
anshul@cs.stonybrook.edu

Erez Zadok
Stony Brook University
Stony Brook, New York, USA
ezk@cs.stonybrook.edu

## ABSTRACT

Sustainability has become a critical focus area across the technology industry, most notably in cloud data centers. In such shared-use computing environments, there is a need to account for the power consumption of individual users. Prior work on power prediction of individual user jobs in shared environments has often focused on workloads that stress a single resource, such as CPU or DRAM. These works typically employ a specific machine learning (ML) model to train and test on the target workload for high accuracy. However, modern workloads in data centers can stress multiple resources simultaneously, and cannot be assumed to always be available for training. This paper empirically evaluates the performance of various ML models under different model settings and training data assumptions for the per-job power prediction problem using a range of workloads. Our evaluation results provide key insights into the efficacy of different ML models. For example, we find that linear ML models suffer from poor prediction accuracy (as much as 25% prediction error), especially for unseen workloads. Conversely, non-linear models, specifically XGBoost and Random Forest, provide reasonable accuracy (7–9% error). We also find that data-normalization and the power-prediction model formulation affect the accuracy of individual ML models in different ways.

## CCS CONCEPTS

• **Hardware → Power and energy; Power estimation and optimization; Enterprise level and data centers power issues**.

## KEYWORDS

Sustainability, per-job power prediction, ML models, co-executed workloads.

**ACM Reference Format:**
Debajyoti Halder, Manas Acharya, Aniket Malsane, Anshul Gandhi, and Erez Zadok. 2024. Empirical Evaluation of ML Models for Per-Job Power Prediction . In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3651418

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651418

## 1 INTRODUCTION

The exponential growth in digital data, coupled with increasing computational demands (*e.g.*, DNN training and crypto mining), has raised significant questions about the carbon footprint of data centers [31]. Both data center providers and users often share a common interest in regulating carbon usage [12, 25]. To regulate carbon usage, an important first step is to track the power consumption of *each* workload (or job, used interchangeably). This per-job power-tracking enables informed decision-making, empowering users to make design choices that align with sustainability goals [15]. Further, in the near future, providers may consider pricing models that partly charge users based on their attributed power use (*e.g.*, carbon tax), thus incentivizing sustainable practices.

Predicting the per-job power consumption is a difficult problem due to the often time-varying utilization of the various server resources by a job at runtime. The problem is further exacerbated by OS- and device-specific scheduling intricacies when resources have to be *shared* between jobs. Machine Learning (ML) approaches, such as regression models, are well suited to the power prediction problem given their ability to infer complex relationships between variables [25]. In particular, prior works have used a variety of ML models and model settings for the power-prediction problem. However, given the large variety of ML models and their settings, *a thorough evaluation is first necessary to assess the usefulness of different ML models for job-level power prediction.*

Recent works on *per-job power prediction* have primarily focused on estimating the power consumption based on CPU and memory utilization metrics [6, 13, 14]. As we discuss in Section 2, it is not enough to account only for the power consumption of CPU and memory subsystems. The classical works in power prediction (*e.g.*, Joulemeter [21], VMeter [17]) employ linear ML models to predict power as a function of resource-utilization metrics. While such models may work well for benchmarks designed to saturate individual resources, we find that linear models have poor accuracy when predicting per-job power for workloads that stress multiple resources simultaneously, such as TensorFlow and MongoDB.

In this paper, we empirically evaluate the performance of several, diverse ML models (both linear and non-linear) to predict per-job power consumption, using several workloads and both micro- and macro-benchmarks. We also evaluate the impact on

prediction accuracy of several models and system settings, such as data normalization, accounting for background processes, and factoring in idle power. To investigate the impact of training data and the deployment context, we evaluate the ML models under different settings, including testing on the training workloads and testing on unseen workloads.

Our experimental results using 8 different pairs of (co-executed) test workloads under 7 different ML models show that *non-linear models outperform linear models* in terms of per-job prediction accuracy (see Section 4). In particular, XGBoost and Random Forest provided less than 10% error when predicting the per-job power consumption of *unseen* workloads (comparing the sum of predicted per-job power values with full-server power measurements). By contrast, linear regression (LR) had much worse accuracy, with errors as high as 40–50% for some pairs of co-located workloads. Our experiments beyond two co-executed workloads show that non-linear models can predict per-job power consumption with ∼10% error when the training dataset corresponds to the test co-execution scenario. However, training a model for each workload class (CPU-, DRAM-, I/O-heavy) separately did not improve the prediction accuracy significantly (3–7% difference).

We also find that the ML model settings and prediction formulation can have an impact on prediction accuracy. For example, predicting for the residual power (after subtracting idle system power) instead of total power, and including the intercept term in supported ML models, can reduce prediction error by as much 10%. We also found that data normalization techniques like standardization or min-max scaling significantly affect neural network models' accuracy. Other models, such as decision trees, are less sensitive to data scaling. Finally, we found that prediction accuracy is not much affected when we take into account the resource utilization of background processes, suggesting that ML models can capture such activities through the resource usage of foreground workloads.

In summary, this paper makes the following contributions:

- We empirically evaluate ***several ML models***, including a workload-specific model, for power prediction. This is in contrast to existing works that often only consider a single model. Further, we report on the impact of model formulation settings and data-processing techniques on power prediction accuracy. We have made our datasets and code available [27] for reproducibility.
- We consider the practical yet challenging problem of ***per-job power prediction*** to allow users in shared environments to assess their sustainability footprint. We considered up to four co-executed workloads. Few prior works focus on this realistic case.
- Unlike prior works, we experiment with ***diverse workloads*** that are not just CPU- or memory-bound but stress the entire system. Further, we consider the realistic scenario where a test workload *has not been observed* in training, unlike much of the prior work that focuses only on cross-validation results.

## 2 BACKGROUND AND PRIOR WORK

The problem of predicting the power consumption of individual jobs *in the presence of other co-executing jobs* is challenging for at least two reasons. First, the power consumption of a server when running multiple jobs simultaneously is not simply the sum of the power consumed when the jobs are run individually (see Appendix A for empirical data). This is likely due to resource saturation, sharing, and contention when multiple jobs co-execute.

The second challenge is that there is no accurate, ground truth power value that is available for *individual jobs*. Prior research has focused on predicting *total* server power [3, 32] by using resource usage metrics (*e.g.*, CPU and memory utilization). While these models are valuable for specific use cases, our goal is to model the concurrent utilization of all system resources to predict *per-job* power consumption.

*Prior Work.* Joulemeter [21] employs linear ML models to predict per-VM power consumption using observable power states in the hypervisor. Linear regression models have also been employed in CloudMonitor [32] and VMeter [17] to predict VM-level power. Likewise, linear regression has also been used to predict total server power (Krishnan *et al.* [23]) and process-level power (Bertran *et al.* [4]). However, we find that linear models are inadequate for predicting the power consumption of co-executed jobs in shared environment scenarios (see Section 4.2).

Several studies have employed a single, non-linear ML model for power prediction. Xiao *et al.* [35] and BitWatts by Colmant *et al.* [7] explore polynomial regression for predicting power in virtual environments. Dhiman *et al.* [11] proposed the use of Gaussian Mixture Models (GMM) for power prediction in virtualized environments. Recent works by Fieni *et al.* [13, 14] leverage Lasso and Ridge regression for power modeling. The authors also use sequential learning to calibrate their power models online by training on currently executing workloads [14]. The authors also rely on PowerAPI [18] (which in turn relies on Intel RAPL [9]) to track CPU package and DRAM power consumption values as ground truth. RAPL-reported values provide power consumption of CPU package (cores, caches, and any integrated GPU) and DRAM. Phung *et al.* [30] leverage RAPL values as additional features for learning, but focus on modeling the power use of only CPU-intensive workloads.

While RAPL power values can serve as ground truth for CPU and/or DRAM power consumption, they are not accurate indicators of full-server power (see Appendix B) as *RAPL does not include power consumed by disks, motherboard, network, or GPU(s).*

Given the importance of power consumption tracking, there have also been power modeling tools developed for consumer use. Scaphandre [28] predicts per-process power consumption by tracking the jiffies and correlating it with RAPL power values when a process is running. However, as noted by prior work [20], Scaphandre's focus is primarily on CPU-power consumption (hence the reliance on RAPL). Kepler [6] is a tool developed by Red Hat that predicts pod and node power consumption; however, Kepler is limited to only Kubernetes environments. Further, Kepler uses cgroups and sysfs to get CPU and memory usage statistics, and thus only focuses on the power of these two resources. Similar tools have also been developed for user-facing purposes, but these tools are not accurate enough for tracking the power of workloads that stress multiple resources. For example, Apple provides an "Energy Impact" metric with their Activity Monitor tool [2], but the estimates reported are only relative values (with no units) that are based on a job's CPU usage [26].

There are other prior works that focused on specific scenarios or workloads (see survey paper by Lin *et al.* [25]), such as approaches

**Table 1: Features used to train ML models.**

| Entity | Features |
|---|---|
| CPU | Cycles, Ref-cycles, Instructions |
| DRAM | LLC-load-misses, LLC-loads, LLC-store-misses, LLC-stores |
| Disk | Bytes, Blocks (# of reads and writes) |
| RAPL | Package power, RAM power |

that estimate the power usage of HPC servers [19, 34] or predict the power consumption of DNN training systems [1, 24, 31]. However, they rely on the specifics of the workload or system, and are thus not easily generalizable.

## 3 POWER MODELING

For all ML models we considered, the ground truth for server power, $P_{server}$, was obtained via a power meter (see Section 4.1). In general, the ML models estimate server power at time $t$, say $\hat{P}^t_{server}$, as a function of some feature vector, $\vec{x}^t$, as $\hat{P}^t_{server} = f(\vec{x}^t)$. The hat notation denotes predicted values (as opposed to ground truth values). For ease of notation, we drop the $t$ superscript by implicitly considering the formulations as being specific to a given time.

### 3.1 Features

The ML models aim to predict power consumption as a function of resource utilization and other metrics, referred to as features (the $\vec{x}$). Rather than determining these features from scratch, we built on existing studies to obtain features for our power-prediction problem; note that we are not considering networked systems in this paper, so we do not include network features, though they could be easily added as needed.

Based on prior works [17, 21, 32], we arrived at the feature list shown in Table 1. We believe this list is short yet representative enough to capture the important resource-utilization values. While RAPL power values may not track full-server power, RAPL power values may still serve as useful *features*, as we explore in Section 4. For CPU and DRAM features, we used `perf-stat` to obtain performance event counts, which are reported per TID (Thread Identifier). We used `pstree` to track all TIDs (including for child threads) pertaining to a given workload to aggregate the features from `perf-stat` *per workload*. We used `blktrace` to track per-process disk reads and writes.

All performance event counts from `perf-stat` are sampled at 200ms intervals. A higher sample rate increased the power consumption overhead of tracing by 5W. We aggregate performance event counts for every 1s interval and align them with the full-system power values obtained from the power meter every 1s. For workload-specific resource utilization, we combine the performance event counts for each workload separately. We obtain RAPL power values from `turbostat`; RAPL power values are reported as an aggregate for the CPU package and DRAM, and not per TID.

### 3.2 Power Prediction

We start with a simple setting where a single workload is running on a server. In this case, the power prediction can be formulated as:

$$\hat{P}_{server} = f(\vec{x_{w1}}) \tag{1}$$

where $\vec{x_{w1}}$ is the feature vector, say of size $n$, obtained for the (single) workload. For example, for Linear Regression (LR) with intercept term, Eq. (1) takes the form $\hat{P}_{server} = \beta_0 + \sum_{i=1}^{n} \beta_i \cdot x_{w1.i}$, where the $\beta$ terms denote the coefficients of the LR model that are learned during training and $x_{w1.i}$ is the $i^{th}$ feature of the $\vec{x_{w1}}$ feature vector. For LR (and other ML models that support the intercept term, $\beta_0$), one can also set up the ML model without intercept. We evaluate both options in our experiments.

Since the server has some baseline idle power, say $P_{idle}$, which can be considered as a constant (for that server), another formulation is to predict the *residual* power (or dynamic power), which is $P^{res}_{server} = P_{server} - P_{idle}$. In this case, the prediction takes the form $\hat{P}^{res}_{server} = f(\vec{x_{w1}})$, and so we predict full-server power as:

$$\hat{P}_{server} = P_{idle} + f(\vec{x_{w1}}) \tag{2}$$

Another variant of Eq. (2) is to also subtract the feature values obtained for an idle system, say $\vec{x_{idle}}$ (*e.g.*, CPU cycles of an idle system spent on background processes) to better correlate with residual power as:

$$\hat{P}_{server} = P_{idle} + f(\vec{x_{w1}} - \vec{x_{idle}}) \tag{3}$$

*Co-Executed Workloads.* When we have two workloads (can be extended beyond two) executing concurrently, as is the focus of this paper, we can separately predict the power consumption of each workload as $\hat{P}_{wi} = f(\vec{x_{wi}})$, and predict full-server power as:

$$\hat{P}_{server} = f(\vec{x_{w1}}) + f(\vec{x_{w2}}) \tag{4}$$

For ML models that have an intercept term (*e.g.*, Linear Regression with $\beta_0$ in $f()$), we subtract the intercept once from Eq. (4) to avoid double-counting the intercept. Note that we still use full-server power ($P_{server}$) as the dependent variable in the final prediction formulation since we have ground truth for only full-server power (and not for the power consumption of individual workloads).

For co-executed scenarios, we separately track the feature values (*e.g.*, CPU cycles) for each workload process and their children to obtain $\vec{x_{w1}}$ and $\vec{x_{w2}}$. Feature values that do not belong to either of the processes can be attributed to background or kernel processes, denoted as $\vec{x_{bg}}$. As such, another variant of Eq. (4) that we consider is with $f(\vec{x_{bg}})$ added to the right-hand side.

For the residual power formulation, we similarly have:

$$\hat{P}_{server} = P_{idle} + f(\vec{x_{w1}}) + f(\vec{x_{w2}}) \tag{5}$$

with the possibility of $f(\vec{x_{bg}})$ added to the right-hand side. We also consider variants of the above formulations where $\vec{x_{idle}}$ is subtracted from each feature vector on the right-hand side.

*Power accounting.* For the co-executed formulations, Eqs. (4) and (5), and their variants, the power contribution of each workload can be estimated as $f(\vec{x_{wi}})$, for $i = 1, 2$. If $P_{idle}$ (or the intercept term or $f(\vec{x_{bg}})$) also must be accounted for, then we can charge each workload with a fraction of $P_{idle}$ proportional to its estimated power. For example, in Eq. (5), we estimate workload 1's total power contribution as:

$$f(\vec{x_{w1}}) + \frac{f(\vec{x_{w1}})}{f(\vec{x_{w1}}) + f(\vec{x_{w2}})} \cdot P_{idle} \tag{6}$$

**Table 2: Workloads employed in our experiments.**

| Resource | Workload |
|---|---|
| CPU | 7zip, Cypto++, CP2K, Gzip |
| DRAM | Stream, MBW, Tinymembench, RAMSpeed SMP |
| Disk | Unpack Linux, LevelDB, SQLite, FIO |
| System | Stress-ng, Tensorflow, Mobile Neural Network, Sysbench, Memcached, Filebench, MongoDB |

Using the proportion of predicted power to account for $P_{idle}$ is preferable to, say, using the proportion of a resource usage metric, since predicted power is a function of all features.

## 4 EVALUATION

In this section we discuss the results and observations from the evaluation of the power models on various workloads and model formulations. In Section 4.1 we discuss our experimental setup, the benchmarks used for evaluation, the ML models, their model formulation settings, and the metrics used for evaluation. We conducted experiments for multiple scenarios like predicting per-job power when only a single workload is executed, or when two or more workloads are co-executed. The evaluation results for every scenario are discussed in Section 4.2. We also discuss 5-fold cross-validation results and feature importance.

To make our results reproducible, we have made available our datasets and code for power-prediction model evaluation [27].

### 4.1 Experimental Setup and Methodology

We conduct all our experiments on a server with two Intel Xeon E5 CPUs with Haswell architecture (has RAPL support). The server has 24 cores total and 256GB of memory. We disabled speedstep (DVFS), hyperthreading, and turboboost (overclocking) to minimize power consumption uncertainties due to dynamic system/OS behavior. To obtain ground truth, we use an external wall power meter, WattsUp Pro [33], attached to the server, which provides full-server power readings once per second.

*Workloads.* For our evaluation, we employed workloads from stress-ng [22], YCSB [8], and Phoronix Test Suite [29], as shown in Table 2. The resource-specific workloads were primarily used for training whereas the System workloads were used for testing; a similar methodology was adopted by prior works that modeled the power consumption of individual (not co-executed) workloads [7, 13, 16]. Training on microbenchmarks allows the ML models to learn the impact of resource utilization on power consumption under controlled stress-test conditions. Every workload ran for around 20 minutes either independently, or co-executed with other workloads.

*ML Models.* We used a variety of ML models to evaluate power prediction: Linear Regression (LR), Decision Tree (DT), Random Forest (RF), Support Vector Regressor (SVR), XGBoost, Lasso, and Neural Network (NN). All ML model hyper-parameters were tuned via Grid Search; see Appendix C for details.

*Data Processing Techniques.* For data processing, we experimented with three popular techniques: (i) de-mean, whereby the mean of the dataset is subtracted, (ii) standardization, which additionally divides by the standard deviation of the data, and (iii) min-max normalization, which scales data to the $(0, 1)$ range. In general, all techniques provided better results than no processing as raw values for different features have different magnitudes. For example, CPU cycles/second is usually in the billions, whereas for non-disk-intensive workloads, the reads/second or writes/second during workload execution is typically in the thousands.

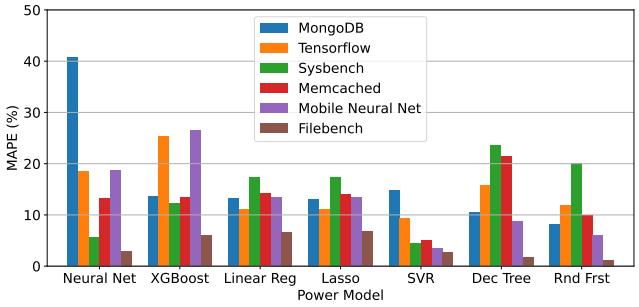



**Figure 1: MAPE values for power modeling when a single application is run on the server.**

*Prediction Metrics.* We used Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) as our error metrics; ground truth was obtained from the power meter. MAE values showed the same trend as MAPE, so we report MAPE values in our results.

### 4.2 Experimental Results

For evaluation, unless otherwise stated, we use the System workloads (as listed in Table 2) for testing the ML models, while using the others for training.

*4.2.1 Single Workload Execution.* We start by evaluating the ML models for the single workload scenario using the residual power prediction formula with idle features removed (see Eq. (3)). Figure 1 illustrates the MAPE values obtained for predicting the server power on the y-axis and the ML models on the x-axis. All the models perform well, with an average MAPE value of less than 17% across all workloads. Support Vector Regression (SVR) outperforms the others, with an average MAPE of 6.6%, followed by Random Forest with an average MAPE of 9.6%. Even Linear Regression (LR) provides satisfactory results, achieving an average MAPE of 12.7%.

For the power prediction in Figure 1, we experimented with different data-processing techniques. We found that the de-mean approach provided the best results for all models, except NN. For all models except NN, other approaches like standardization result in a 1–2% increase in MAPE. With min-max normalization, MAPE increased by 1–3%. For NN, min-max scaling worked the best, improving the prediction accuracy significantly (76%); standardization only provided some improvement (11%) in prediction accuracy. The reason for this is the NN model's sensitivity to input scale and reliance on gradient-based optimization methods [5]. Techniques like min-max scaling ensure a consistent scale for all features, facilitating efficient learning and stable convergence. Without this consistent scaling, the NN model's learning could be inefficient due to skewed gradients. Other models such as Decision trees, Random Forest, and XGBoost, are less sensitive to data scaling because their splitting criteria and/or ensemble nature focuses on the relative ordering of feature values rather than their specific scales. These models make decisions based on feature relationships, making them

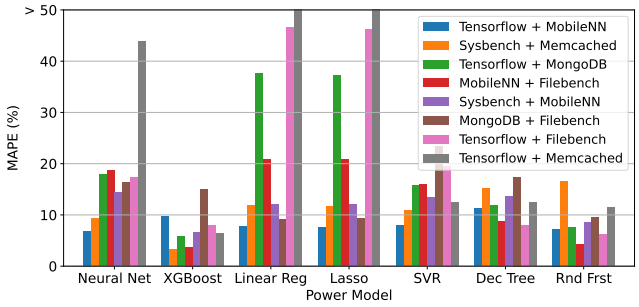



**Figure 2: Prediction results when workloads are co-executed and residual server power is predicted as the sum of predicted residual power of each workload.**

inherently robust to variations in feature scales. For subsequent evaluations, we used de-mean for all models except NN, for which we used min-max scaling.

Among the different model variants considered, the residual power prediction approach (Eq. (2)) offered better accuracy compared to directly predicting full-server power (Eq. (1)). This improvement is seen particularly in the case of Linear Regression (LR) and XGBoost, resulting in a 10% reduction in MAPE. Excluding idle features ($x_{\vec{idle}}$) from the prediction process did not affect prediction accuracy. Furthermore, including the intercept term in supported models led to a slight improvement in prediction accuracy. However, for LR, the improvement was substantial, reducing MAPE by approximately 20%. Unless specified otherwise, we considered these variations in our subsequent results.

Predicting residual power, where we isolate the active system usage by subtracting idle power, proved effective in improving accuracy. This is because predicting residual power allows the model to capture only the specific resource patterns associated with active jobs, providing a better understanding of how power usage is affected by resource utilization of jobs. At the same time, incorporating the intercept term in the models was crucial for considering baseline power, representing the constant power consumption of the server (which includes idle power) when no active jobs are running. *These two steps (predicting for residual power and including the intercept term) are important for power prediction as they ensure that the model does not unintentionally miss or attribute variations, preventing bias in predictions.*

Additionally, we conducted experiments without utilizing the two RAPL power features. This resulted in a slight increase (1-2%) in MAPE values across all ML models. RAPL power features cannot be obtained on a per-thread or Thread ID (TID) basis. Therefore, it is not possible to track the power contribution of individual workloads using RAPL power features. Moreover, RAPL values exhibit inconsistencies for certain server models, as reported by Desrochers *et al.* [10]. Therefore, we opted not to incorporate RAPL power features in the remainder of our evaluation.

*4.2.2 Per-job Power Prediction for Co-Executed Workloads.* We now consider the challenging case where each co-executed workload's power consumption is to be predicted. In particular, each co-executed workload's residual power consumption is first predicted, and then the full-server power, obtained by adding the individual workload power values and $P_{idle}$ (via Eq. (5)), is compared with the full-server ground truth power use. We train our models on feature vectors from random pairs of non-System workloads from Table 2. We then test the models on 8 pairs of System workloads. This ensures that test workloads are separate from training.

Figure 2 shows our results for different pairs of co-executed workloads. Here, the training data included only pairs of *non-test* workloads from Table 2, representing the realistic case where test workloads may not always be available for training. XGBoost performed the best, with an average MAPE of 7.3%, followed by Random Forest (8.9%), Decision Tree (12.3%), and SVR (14.9%). LR performed poorly, with an average MAPE of 25%, highlighting the *linear model's inability to account for resource contention when workloads are co-executed.* We also explored the variation where the predicted power consumed by other processes ($f(\vec{x_{bg}})$ term from Section 3.2) was added to the predicted power of the workloads to arrive at the full-server power prediction. However, this formulation yielded slightly higher MAPE values, and was thus not considered further.

We also conducted 5-fold cross-validation for our ML models by training and testing on the dataset obtained by co-executing a pair of workloads. In general, the prediction errors are lower than those in Figure 2, since the test workloads comprise the same training data. Based on average MAPE values, all models performed well, with SVR (4.4%), RF (5.7%), NN (6.7%), Lasso (6.7%), DT (8.1%), and XGBoost (9.7%) providing less than 10% error; even LR (6.9%) resulted in low error under this cross-validation setting. *This suggests that the choice of ML model to employ also depends on the training and test data overlap assumptions.*

*4.2.3 Background Processes as Third Workload.* As mentioned earlier, we explored another variation of our power model by considering all background processes as an additional workload co-executed during the experiments. The idea behind this approach was that the power model might be able to better segregate the power among the active workloads if the background processes are grouped separately. The power prediction formulation hence becomes:

$$\hat{P}_{server} = P_{idle} + f(\vec{x_{w1}}) + f(\vec{x_{w2}}) + f(\vec{x_{bg}}) \quad (7)$$

However, the results were not impressive. XGBoost had the least average MAPE of 20.9%, followed by RF (22.9%), NN (25.3%), and DT (25.4%). We also tried another variation of this model without the intercept term ($\beta_0$), but that resulted in much worse results (~57% average MAPE for RF, DT, and LR). We thus decided to not consider this model variation for our evaluation.

*4.2.4 Beyond Two Co-Executed Workloads.* We next experimented by testing on four co-executed workloads by also training on the dataset obtained by co-executing four workloads. We ran 5 combinations of four workload pairs and evaluated using "leave-one-out" cross-validation (see Figure 3). For example, in Figure 3, when we test on the Tensorflow + MNN + Memcached + Filebench workload combination (blue legend), then the remaining 4 workload combinations are used for training.

All the ML models performed well except LR and Lasso. XGBoost and Random Forest had the least average MAPE of ~10%. LR performed much worse with ~24% average MAPE. This again shows that *linear models are not well suited for co-executed workloads' power prediction.* Overall, *XGBoost and Random Forest are the*

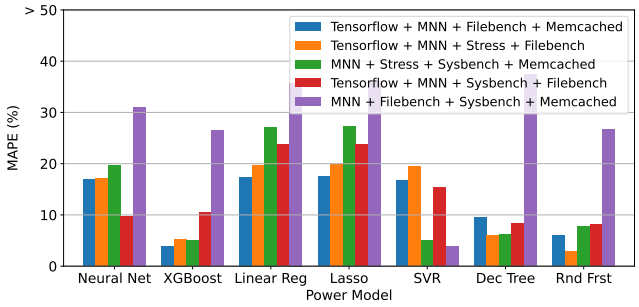



**Figure 3: Prediction results when four workloads are co-executed and residual server power is predicted as the sum of predicted residual power of each workload.**

*best performing models* with 10% or lower average MAPE in most scenarios, showing that non-linear models are effective for per-job power prediction.

*4.2.5 Workload Classification.* The workloads being executed on a server can be classified by the resource(s) they stress. It may be interesting to consider a modeling approach whereby we build a power model for each workload class separately to gain accuracy. To that end, we classified workloads into 3 classes: CPU-heavy, DRAM-heavy, and Disk-heavy. We trained a different power model for each class and tested it on a workload of the same class. We compared this new approach of "workload classification" with the original "all workloads for training" approach (except the one being tested). In this experiment, we trained and tested on a single workload execution scenario. The benchmark workloads used are the micro-benchmarks from Table 2. Figure 4 shows that the prediction error of the power model trained for specific workload classes is typically worse than the original approach (3–7% difference). *This suggests that a single model trained on multiple workload classes can improve power prediction accuracy over workload-specific models.*

*4.2.6 Feature Selection and Importance.* To evaluate our feature set, we employed the `mutual information` method from scikit-learn to estimate the significance of each feature utilized in our training, as outlined in Table 1. This method quantifies the dependence between two variables and is instrumental in assessing the information gain associated with each feature relative to the target variable. We found that DRAM and CPU features had higher importance scores (*e.g.*, LLC-loads, LLC-stores for DRAM and ref-cycles, cycles for CPU), while Disk features (*e.g.*, bytes and blocks) had the lowest scores. We repeated our power predictions by omitting the Disk features, but this resulted in slightly higher MAPE values, suggesting that our feature list is adequate. We also utilized XGBoost and Random Forest algorithms to determine feature importance, obtaining similar results.

*4.2.7 Analyzing the Per-Job Power Predictions.* Thus far we evaluated our per-job power predictions (obtained via Eq. (6)) by comparing the *sum* of per-job powers with full-server power meter readings because there is *no ground truth* for per-job power consumption in the co-executed setting. Nonetheless, we can analyze the trend in per-job power predictions. We first compared the per-job predictions obtained from the co-executed setting with the ground truth residual power when the workload is run in isolation. As expected,

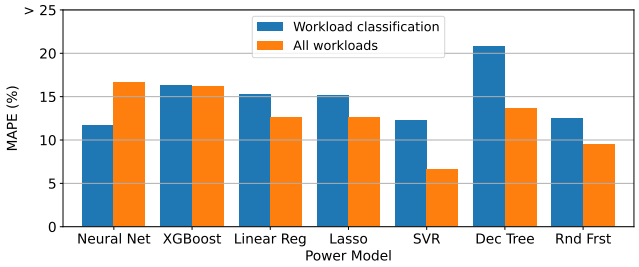



**Figure 4: Performance of power models when trained on specific workload classes versus all workloads.**

the former is lower than the latter, likely due to resource contention and saturation. For example, the ground truth residual power of Sysbench in isolation is about 63W. However, the predicted per-job power of Sysbench when co-executed with Memcached and when co-executed with Mobile Neural Network (MNN) is only about 46W and 50W, respectively. In both co-executions, the predicted power of Memcached and MNN is also lower than their ground truth isolated power.

We also compared the per-job power predictions when TensorFlow (TF) is co-executed with MongoDB and when TF is co-executed with MNN. The predicted power for TF is 67W when co-executed with MongoDB and 85W when co-executed with MNN. Since MongoDB is I/O intensive, we expect TF's I/O to be slowed down much more when TF is co-executed with MongoDB compared to when TF is co-executed with MNN; consequently, TF may have fewer instructions to be run per second when co-executed with MongoDB, lowering its power draw. The disk and CPU features confirm this claim, providing some validation of our predictions.

## 5 CONCLUSION AND FUTURE WORK

Per-job power tracking is an important first step for incentivizing sustainable computing practices among consumers and providers of cloud data centers. While there is ample literature on power-prediction techniques, there is little prior work on *comparing different power-prediction models, especially under different workload and model settings*. Our evaluation results showed that non-linear ML models, specifically XGBoost and Random Forest, provided good prediction accuracy ($\leq$ 10% MAPE). Even for SVR models, we found that a non-linear kernel provided significantly higher prediction accuracy than a linear kernel (~92% lower MAPE). By contrast, LR did not perform well (25% MAPE). As such, the choice of ML model plays an important role in power prediction. The choice of ML model also depends on the training and test data assumptions. In cross-validation settings, almost all ML models we experimented with performed quite well (less than 10% MAPE). Interestingly, workload-specific power models did not provide good accuracy; training across workload classes resulted in a non-trivial 6% accuracy gain.

## ACKNOWLEDGMENTS

This work was supported in part by Dell-EMC, NetApp, Tintri, Facebook, and IBM support; and NSF awards CNS-1750109, CNS-1900706, CNS-2106263, CNS-2106434, CNS-2214980, and CCF-2324859.

## REFERENCES


[1] Anthony, L., Kanding, B., and Selvan, R. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. In *ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems* (2020).

[2] Apple. View energy consumption in Activity Monitor on Mac. https://support.apple.com/en-gb/guide/activity-monitor/actmntr43697/mac.

[3] Barroso, L. A., and Hölzle, U. The Case for Energy-Proportional Computing. *IEEE Computer 40*, 12 (2007), 33–37.

[4] Bertran, R., Becerra, Y., Carrera, D., Beltran, V., Gonzalez, M., Martorell, X., Torres, J., and Ayguade, E. Accurate energy accounting for shared virtualized environments using pmc-based power modeling techniques. In *2010 11th IEEE/ACM International Conference on Grid Computing* (2010), pp. 1–8.

[5] Bhanja, S., and Das, A. Impact of data normalization on deep neural network for time series forecasting. *ArXiv* (2018).

[6] Cloud Native Computing Foundation. Kubernetes Efficient Power Level Exporter (Kepler). https://sustainable-computing.io, 2022.

[7] Colmant, M., Kurpicz, M., Felber, P., Huertas, L., Rouvoy, R., and Sobe, A. Process-level power estimation in vm-based systems. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, Association for Computing Machinery.

[8] Cooper, Brian. Yahoo! Cloud Serving Benchmark. https://github.com/brianfrankcooper/YCSB, 2021.

[9] David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R., and Le, C. RAPL: memory power estimation and capping. In *Proceedings of the 2010 ACM/IEEE International Symposium on Low Power Electronics and Design (ISPLED)* (2010), pp. 189–194.

[10] Desrochers, S., Paradis, C., and Weaver, V. M. A Validation of DRAM RAPL Power Measurements. In *Proceedings of the Second International Symposium on Memory Systems* (Alexandria, VA, USA, 2016), pp. 455–470.

[11] Dhiman, G., Mihic, K., and Rosing, T. A system for online power prediction in virtualized environments using gaussian mixture models. In *Design Automation Conference* (2010), pp. 807–812.

[12] Dutt, A., Rachuri, S. P., Lobo, A., Shaik, N., Gandhi, A., and Liu, Z. Evaluating the energy impact of device parameters for dnn inference on edge. In *Proceedings of the 14th International Green and Sustainable Computing Conference (IGSC'23)* (Toronto, Canada, 2023).

[13] Fieni, G., Rouvoy, R., and Seinturier, L. SmartWatts: Self-Calibrating Software-Defined Power Meter for Containers. In *Proceedings of the 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing* (2020), pp. 479–488.

[14] Fieni, G., Rouvoy, R., and Seiturier, L. SelfWatts: On-the-fly Selection of Performance Events to Optimize Software-defined Power Meters. In *Proceedings of the 21st International Symposium on Cluster, Cloud and Internet Computing* (2021), pp. 324–333.

[15] Gandhi, A., Ghose, K., Gopalan, K., Hussain, S., Lee, D., Liu, D., Liu, Z., McDaniel, P., Mu, S., and Zadok, E. Metrics for sustainability in data centers. In *Proceedings of the 1st Workshop on Sustainable Computer Systems Design and Implementation (HotCarbon'22)* (San Diego, CA, USA, July 2022), USENIX.

[16] Guo, N., Gui, W., Chen, W., Tian, X., Qiu, W., Tian, Z., and Zhang, X. Using improved support vector regression to predict the transmitted energy consumption data by distributed wireless sensor network. *EURASIP Journal on Wireless Communications and Networking 2020*, 1 (2020), 120.

[17] Husain Bohra, A. E., and Chaudhary, V. VMeter: Power modelling for virtualized clouds. In *Proceedings of the 2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)* (2010), pp. 1–8.

[18] Inria, University of Lille. PowerAPI. https://powerapi.org, 2023.

[19] Jarus, M., Oleksiak, A., Piontek, T., and Weglarz, J. Runtime power usage estimation of HPC servers for various classes of real-life applications. *Future Generation Computer Systems 36* (2014), 299–310.

[20] Jay, M., Ostapenco, V., Lefèvre, L., Trystram, D., Orgerie, A.-C., and Fichel, B. An experimental comparison of software-based power meters: focus on CPU and GPU. In *Proceedings of the 23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing* (Bangalore, India, 2023), pp. 1–13.

[21] Kansal, A., Zhao, F., Liu, J., Kothari, N., and Bhattacharya, A. A. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing* (Indianapolis, IN, USA, 2010), SoCC '10, pp. 39–50.

[22] King, Colin. stress-ng. https://manpages.ubuntu.com/manpages/xenial/man1/stress-ng.1.html, 2023.

[23] Krishnan, B., Amur, H., Gavrilovska, A., and Schwan, K. Vm power metering: Feasibility and challenges. *SIGMETRICS Perform. Eval. Rev. 38*, 3 (jan 2011), 56–60.

[24] Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700* (2019).

[25] Lin, W., Shi, F., Wu, W., Li, K., Wu, G., and Mohammed, A.-A. A taxonomy and survey of power models and power modeling for cloud servers. *ACM Comput. Surv. 53*, 5 (2020).

[26] Nethercote, Nicholas. What does the OS X Activity Monitor's "Energy Impact" actually measure? https://blog.mozilla.org/nnethercote/2015/08/26/what-does-the-os-x-activity-monitors-energy-impact-actually-measure/.

[27] PACE Lab, Stony Brook University. Replication Package. https://github.com/PACELab/sassy-metrics-data-code, 2023.

[28] Petit, B. Scaphandre. https://github.com/hubblo-org/scaphandre.

[29] Phoronix Media. Phoronix Test Suite. https://www.phoronix-test-suite.com/, 2023.

[30] Phung, J., Lee, Y. C., and Zomaya, A. Y. Modeling System-Level Power Consumption Profiles Using RAPL. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)* (2018), pp. 1–4.

[31] Schmidt, V., Goyal, K., Joshi, A., Feld, B., Conell, L., Laskaris, N., Blank, D., Wilson, J., Friedler, S., and Luccioni, S. CodeCarbon: Estimate and Track Carbon Emissions from Machine Learning Computing. https://mlco2.github.io/codecarbon/motivation.html, 2021.

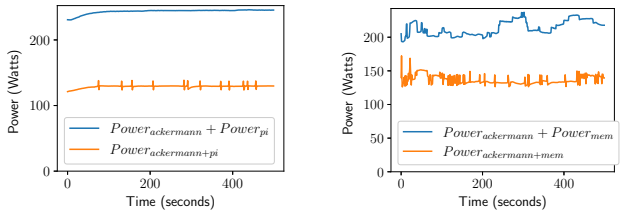
[32] Smith, J. W., Khajeh-Hosseini, A., Ward, J. S., and Sommerville, I. Cloudmonitor: Profiling power usage. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing* (2012), pp. 947–948.

[33] WattsUp. WattsUp? Pro. https://arcb.csc.ncsu.edu/~mueller/cluster/arc/wattsup/metertools-1.0.0/docs/meters/wattsup/manual.pdf.

[34] Witkowski, M., Oleksiak, A., Piontek, T., and Wundefinedglarz, J. Practical Power Consumption Estimation for Real Life HPC Applications. *Future Generation Computer Systems 29*, 1 (2013), 208–217.

[35] Xiao, P., Hu, Z., Liu, D., Yan, G., and Qu, X. Virtual machine power measuring technique with bounded error in cloud environments. *Journal of Network and Computer Applications 36*, 2 (2013), 818–828.


# A APPENDIX: EMPIRICAL DATA SHOWING THE POWER CONSUMPTION OF CO-EXECUTED WORKLOADS VERSUS THE SUM OF POWER CONSUMPTION OF INDIVIDUAL WORKLOADS




(a) CPU load with CPU load (b) CPU load with MEM load

**Figure 5: Residual power (after subtracting idle power) when workloads are co-executed versus sum of residual powers when workloads are run in isolation.**

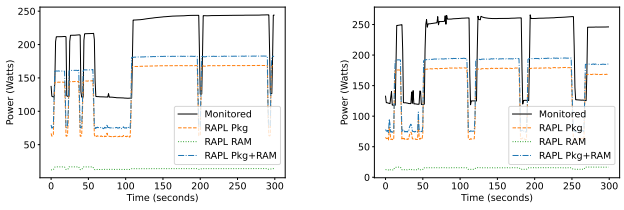
We conducted experiments using stress-ng micro-benchmarks [22] to investigate the difference in power consumption of co-executed and individual workloads. We observed a large difference (see Figure 5(a)) between the sum of residual power consumption of two CPU-bound micro-benchmarks (ackermann and pi) when run individually and the residual power consumption when these two micro-benchmarks are run together. Residual power is the server power with idle power subtracted from it.

Figure 5(a) shows, in blue, the sum of residual power consumption of two CPU-bound micro-benchmarks (ackermann and pi) when run individually; in orange, we see the residual power consumption when these two micro-benchmarks are run together. We see a similar difference (see Figure 5(b)) even if we run a CPU-bound micro-benchmark next to a memory-bound one. The large difference between the two lines shows that the power consumption profile of a job depends on other concurrent jobs as there may be resource saturation and contention when jobs co-execute.

**Table 3: Illustration of our hyper-parameter tuning using Grid Search. Tuned values are highlighted in bold.**

| XGBoost | SVR | Decision Tree | Random Forest | Neural Network |
|---|---|---|---|---|
| learning_rate: [0.01, **0.03**, 0.1, 0.5]<br>n_estimators: [100, 200, 300, **900**]<br>max_depth: [3, 5, **6**]<br>min_child_weight: [1, **3**, 5]<br>subsample: [**0.6**, 0.8, 1.0]<br>colsample_bytree: [**0.6**, 0.8, 1.0] | C: [0.1, **1**, 10, 100]<br>kernel: [linear, **rbf**, poly]<br>gamma: [**scale**, auto, 0.01, 0.1, 1]<br>epsilon: [0.1, 0.2, **0.5**, 1.0] | max_depth: [None, 5, 10, 15, **20**]<br>min_samples_split: [**2**, 5, 10]<br>min_samples_leaf: [1, **2**, 4]<br>max_features: [auto, sqrt, **log2**]<br>max_leaf_nodes: [None, **10**, 20, 30]<br>min_impurity_decrease: [**0.0**, 0.1, 0.2] | n_estimators: [100, 200, **300**, 500]<br>max_depth: [None, 5, 10, **20**, 30]<br>min_samples_split: [**2**, 5, 10]<br>min_samples_leaf: [1, **2**, 4]<br>max_features: ['auto', **'sqrt'**, 'log2']<br>bootstrap: [**True**, False]<br>max_leaf_nodes: [None, 10, 20, 50] | activation: [**ReLU**, ELU, tanh]<br>solver: [adam, **sgd**]<br>learning_rate_init: [0.001, 0.01, **0.1**, 1] |

# B APPENDIX: EMPIRICAL DATA SHOWING THE DIFFERENCE BETWEEN FULL-SERVER MONITORED POWER AND RAPL POWER VALUES




(a) Sysbench (b) Memcached

**Figure 6: Power values reported by Intel RAPL [9] and the full-server power meter when running (a) Sysbench, and (b) Memcached.**

While RAPL power values can serve as ground truth for CPU and/or DRAM power consumption, they are not accurate indicators of full-server power, as shown in Figure 6. We empirically show this shortcoming with Sysbench and Memcached workloads. In general, across workloads, we found that the sum of CPU package and DRAM power values for RAPL is 30–50% lower than full-server monitored power values. Further, within a workload execution, the ratio of full-server to RAPL power values varies significantly, by as much as 1.1–1.7× for the workloads we experimented with. This is to be expected as RAPL does not include power consumed by, for example, the disks, motherboard, network, or non-integrated GPU(s).

# C APPENDIX: HYPER-PARAMETER TUNING

All ML models we experimented with were first tuned via Grid Search to identify the best hyper-parameter values. Table 3 shows the hyper-parameter tuning details for five ML models (XGBoost, SVR, DT, RF, and NN), along with the best values chosen. For XGBoost, parameters such as learning rate, number of estimators, and maximum depth were adjusted to find a balance between model complexity and accuracy. SVR tuning focused on the regularization parameter C, kernel type, kernel coefficient (gamma), and epsilon (for epsilon-SVR model). For kernel type, our experiments showed that the RBF (Radial Basis Function) kernel had the best prediction accuracy. RF and DT had similar tuning to prevent overfitting while maintaining model depth. For RF, we also considered the number of estimators (300 in our case) for better accuracy. For NN, we used the Multi-layer Perceptron regressor from scikit-learn with ReLU activation and three hidden layers; the three hidden layers had size of 512, 16, and 16 neurons. For LR, we experimented with and without intercept. Including intercept, as mentioned earlier, gave better results. We also tried Ridge regression as an alternative to LR and Lasso, but its prediction accuracy was worse (5–6% higher MAPE than LR and Lasso), so we do not include it in our evaluation. For regularization, since Ridge regression did not work well, we had the alpha hyper-parameter set to 0.1 for Lasso (L1 regularization).

# FootPrinter: Quantifying Data Center Carbon Footprint

Dante Niewenhuis
d.niewenhuis@vu.nl
Vrije Universiteit Amsterdam
Amsterdam, Netherlands

Sacheendra Talluri
s.talluri@vu.nl
Vrije Universiteit Amsterdam
Amsterdam, Netherlands

Alexandru Iosup
a.iosup@vu.nl
Vrije Universiteit Amsterdam
Amsterdam, Netherlands

Tiziano De Matteis
t.de.matteis@vu.nl
Vrije Universiteit Amsterdam
Amsterdam, Netherlands

## ABSTRACT

Data centers have become an increasingly significant contributor to the global carbon footprint. In 2021, the global data center industry was responsible for around 1% of the worldwide greenhouse gas emissions. With more resource-intensive workloads, such as Large Language Models, gaining popularity, this percentage is expected to increase further. Therefore, it is crucial for data center service providers to become aware of and accountable for the sustainability impact of their design and operational choices. However, reducing the carbon footprint of data centers has been a challenging process due to the lack of comprehensive metrics, carbon-aware design tools, and guidelines for carbon-aware optimization. In this work, we propose FootPrinter, a first-of-its-kind tool that supports data center designers and operators in assessing the environmental impact of their data center. FootPrinter uses coarse-grained operational data, grid energy mix information, and discrete event simulation to determine the data center's operational carbon footprint and evaluate the impact of infrastructural or operational changes. FootPrinter can simulate days of operations of a regional data center on a commodity laptop in a few seconds, returning the estimated footprint with marginal error. By making this project open source, we hope to engage the community in the development of methodologies and tools for systematically assessing and exploring the sustainability of data centers.

## CCS CONCEPTS

• **Hardware → Impact on the environment**; **Renewable energy**; • **Computing methodologies → Agent / discrete models**.

## KEYWORDS

Carbon Footprint, Carbon Emission, Data Center, Simulation

**ACM Reference Format:**
Dante Niewenhuis, Sacheendra Talluri, Alexandru Iosup, and Tiziano De Matteis. 2024. FootPrinter: Quantifying Data Center Carbon Footprint. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651419

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651419

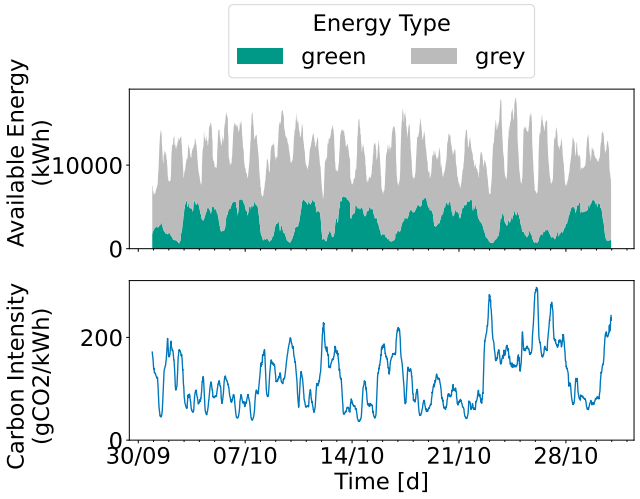



**Figure 1: The energy mix and carbon intensity of the energy grid in the Netherlands during the month of October 2023 from the ENTSO-E Transparency Platform[1]. The top graph shows the energy mix during the month into green and non-green energy. The bottom graph shows the resulting carbon intensity of the grid.**

## 1 INTRODUCTION

Climate change is a significant social challenge today, affecting various aspects of our daily lives [34]. In 2015, world leaders reached a breakthrough with the Paris Agreement, which aims "to limit the temperature increase to 1.5°C above pre-industrial levels." [31]. To achieve this goal, the European Union (EU) has established a 55% reduction in greenhouse gas emissions by 2030 for all its member states [12].

Data centers significantly contribute to the global carbon footprint [13], accounting for 1% of global greenhouse gas emissions in 2021 [24]. As a result of demands from governments and users [30], and financial considerations, data center operators have been working to reduce their carbon footprint. The recent energy price crisis and sustainability efforts (e.g., through green bond emissions) have made operational expenses a primary cost factor for data centers [36].

[1] https://transparency.entsoe.eu/dashboard/show

So far, data center designers and operators have been focusing mainly on improving their power efficiency. Data centers already use more than 1% of the global energy consumption [28], and some estimate this will rise to as much as 8% in 2030 [3]. Despite the improvements in energy efficiency, aggregate energy usage has increased in the last 15 years [7]. Moreover, efficiency improvements have slowed down significantly in recent years [38]. A bigger problem, however, is that optimizing energy does not directly reduce the carbon footprint. The carbon emitted by a data center depends not just on the amount of energy used but also on the type. For instance, Figure 1 shows how the grid energy mix and its carbon intensity can change continuously over time.

Reducing the carbon footprint of a data center is a challenging process. There is no consensus on measuring carbon emissions [19], and *there is a lack of carbon-aware design tools and guidelines for carbon-aware optimization* [18]. These challenges have resulted in many companies still relying on rule-of-thumb reasoning [4], which has led to carbon-inefficient practices, such as significant overprovisioning of resources [20]. Improving the carbon footprint has been even more difficult for smaller data centers [23], which often lack insight into tenant workloads and their provided energy mix. Besides the technical challenges, significant costs are involved. Data centers operate on a large scale, making experimentation costly and time-consuming. Making uninformed decisions can also have a significant economic impact. Data center projects have been stopped in countries like the Netherlands based on vague, qualitative statements about their potential climate impact[2].
In this work, we make three contributions:

(1) *We discuss what information data center operators need to quantify and optimize their operational carbon footprint.* Measuring a data center's energy consumption requires that operators invest in hardware and software tools. Attributing this to individual applications is complex and requires even more tooling. Therefore, we suggest using coarse-grained execution metrics, as a convenient yet effective way of assessing the data center's energy consumption.
(2) *We introduce FootPrinter*[3], a data center discrete simulator based on the OpenDC[4] framework. FootPrinter takes as input the hardware configuration of a data center and workload traces and uses simulation to determine the corresponding energy footprint. The energy profile is combined with the energy mix of the location region to calculate the operational carbon footprint of the data center when it runs the given workload.
(3) *We validate FootPrinter* using a wall-socket energy trace from SURF, the Dutch national supercomputing center, showing that the simulated data center has the same energy usage as the data center running the same workload in the real world.

With FootPrinter, we aim to contribute with a tool for data center designers and operators to reason about the environmental impact and associated costs of their infrastructures and plan for appropriate measures to improve their sustainability.

[2]https://www.datacenterknowledge.com/meta-facebook/scorned-meta-data-center-holland-met-all-environmental-standards
[3]https://github.com/atlarge-research/FootPrinter
[4]https://opendc.org/

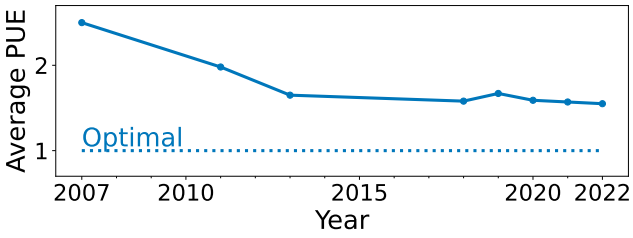



**Figure 2: The average Power Usage Effectiveness (PUE) of 669 data centers from 2007 to 2022 [14]. The dotted line shows the optimal value of 1.0.**

## 2 BACKGROUND

The carbon footprint of a data center is characterized by two types of carbon emission: the *embodied* carbon footprint and the *operational* carbon footprint. Embodied carbon is the carbon emitted from manufacturing and production. Operational carbon footprint is the CO2 emissions caused by energy usage during operations. In this work, we focus on reducing the operational carbon footprint of data centers.

### 2.1 Power Usage Effectiveness

In recent years, much focus has been placed on improving the efficiency of data centers. The most commonly used metric for energy efficiency is Power Usage Effectiveness (PUE). PUE is calculated using Equation 1:

$$PUE = \frac{E_T}{E_{IT}} \quad (1)$$

In which $E_t$ and $E_{IT}$ denote the total energy used by the data center and the energy used by the IT components of the data center. In an optimal data center, no energy is required for redundant tasks, using all energy for the IT equipment doing the computation. This results in a PUE of 1.0. However, while many data centers have been able to optimize their PUE, with for instance Google getting close to 1.1 [5], the aggregate energy consumption of data centers has still increased over the last 15 years [7]. One reason for this is the rebound effect, which states that if the energy required to perform a task (and thus its price) decreases, the number of tasks performed will increase [40]. Another reason is that the rate of improvement of PUE has slowed down significantly in recent years. Figure 2 shows the average PUE of 669 data centers during the period of 2007 to 2022 [14]. While great improvements were made between 2007 and 2013 (from 2.5 to 1.6), recent years did not bring any more significant improvements, with the lowest average PUE of 1.55 being achieved in 2022.

We suggest two possible reasons for this slowdown of improvement. First, as the PUE is already highly optimized, it is becoming increasingly difficult to optimize it further. Second, the shift to hyperscale data centers had a significant impact on the average PUE. Because this shift is nearly finished, it is unclear where significant improvements will come from [7].

[5]https://www.google.com/about/datacenters/efficiency/

## 2.2 Carbon Intensity

While PUE is a good metric to determine infrastructure energy efficiency, it is not taking everything into account. PUE does not consider the energy efficiency of applications and workloads [43]. PUE also completely ignores the type of energy used. The source of energy can have an enormous impact on the carbon emitted. In some cases, energy sourced from renewable sources, such as wind or solar, can emit up to 20x less CO2 compared to traditional energy sources, such as coal [22]. The Carbon Intensity of an energy source defines the amount of carbon emitted per unit of energy used. Many data centers do, however, not use energy from a single energy source, but get their energy from the grid. Energy provided by the grid is often gathered from many different energy sources with different carbon intensities. The carbon intensity of the grid is calculated by aggregating the different energy sources in Equation 2:

$$CI_g = \sum_{s \in S} CI_s \frac{E_s}{E_g} \tag{2}$$

In which $CI_s$ is the carbon intensity of energy source $s$, $E_s/E_g$ is the share of energy that $s$ contributes to the grid, and $S$ is the set of all available energy sources. Green energy is primarily gained from natural phenomena, such as wind or sunlight. This results in a continuously changing mix of available energy (see Figure 1). During this time, the ratio of green and non-green energy varied significantly. As a result, the carbon intensity of the grid also changes significantly over time (100 to 400 gCO2/kWh). This means that to minimize the carbon footprint of a data center, not only the amount of energy used is important, but also when this energy is used.

## 2.3 Operational Footprint

The operational carbon footprint is characterized by the carbon emitted when the system is running. The operational carbon footprint can be calculated by combining the carbon intensity of the data center $CI_d$(gCO2/kWh) and the operational energy of the data center $E_{op}$ (kWh) as defined in Equation 3:

$$C_{op} = CI_d E_{op} \tag{3}$$

We assume that the carbon intensity of the energy used by a data center is proportional to the carbon intensity of the grid ($CI_d = CI_g$). Some data centers have special energy contracts providing them direct access to specific types of energy[6]. However, these data centers still have to resort to using energy from the grid, when not enough energy is available [1]. In this work, we focus on the carbon footprint of a data center. However, several other metrics for data center sustainability exist [35].

## 2.4 Simulation

FootPrinter uses discrete-time simulation to estimate the carbon footprint of a data center in a time and energy-aware manner. Using simulation for data center research is not new. Simulators such as Grid/CloudSim [9], SimGrid [10], and iCanCloud [32] have demonstrated the ability to simulate complex operations at cluster and data center levels. In this work, we use OpenDC, a trace-based discrete data center simulation framework [29]. OpenDC uses real-world workload traces to drive simulation. A workload trace describes when jobs get submitted and their computational requirements. More advanced workload traces also define their computational demand over time. OpenDC replays the workload on a specified data center and allows users to explore "what-if" scenarios. FootPrinter uses these features and extends them to compute the energy required to run the workload on a user-specified data center and derive its corresponding carbon footprint.

[6]https://www.datacenterdynamics.com/en/news/meta-signs-renewable-energy-deal-in-arizona-with-orsted/

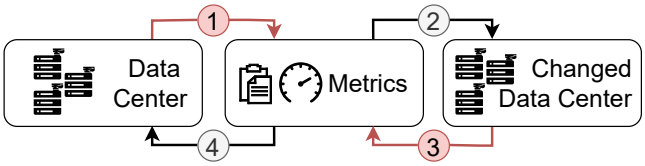



**Figure 3: A method of determining the impact of making changes to a data center. 1) Determine initial performance, 2) Change data center infrastructure and/or operations based on metrics and goals, 3) Determine the performance of the changed data center, and when requirements are met 4) Consolidate the changes in the data center. The red lines highlight the challenging steps.**

# 3 PROBLEM STATEMENT

Reducing the carbon footprint of a data center is a challenging task. Due to a lack of carbon-aware tooling, data center designers and operators need to decide between different options with limited insight into their effects [18]. Therefore, determining how to change the data center infrastructure and operations is often a process of trial and error, in which new experiments are executed based on the results of previous experimentation until the imposed requirements are met (see Figure 3). Using a similar approach when working with data centers is ineffective due to the time, energy, and monetary costs involved. Collecting energy metrics on the level of individual servers or server components requires significant investments in hardware and software, such as power meters for measurement and software to process data and storage. The more detailed the information required, the more power meters, storage, and computing are needed. Furthermore, the energy usage of a data center can assist the operator in identifying problems and areas of improvement, such as idle VMs, or inefficient resource management. It does, however, not provide enough information to determine the effect of changes made to address the identified problems. This insight is vital to determine where to invest the available budget and engineering time. Small real-world experiments followed by analysis are often used to quantify efficacy (see Figure 3). However, this feedback loop might be slow because of the long execution time of experiments, or even unfeasible due to economic reasons.

FootPrinter enables a convenient approach to analyzing and optimizing the carbon footprint of a data center. Through the use of discrete simulations, it allows the user to consider several scenarios, keeping costs and operational impact low. FootPrinters' stakeholders are *data center designers*, who architect the data center infrastructure, and *operators* who run the data center operations.

We present three use cases that showcase the difficulties faced by these stakeholders. In the remainder of the paper, we elaborate on how FootPrinter can currently be utilized to tackle the first two, while *UC-Hardware* is utilized to discuss how FootPrinter's capabilities can be expanded.

***UC-Footprint*** *Operational carbon footprint:* Knowing the operational carbon footprint of a data center is an essential part of evaluating its effectiveness. Determining the operational carbon footprint requires knowledge about both the energy usage and the carbon intensity of the used energy sources. As discussed previously, properly monitoring energy usage requires specialized hardware and software.

***UC-Location*** *Selecting a location:* The location of a data center can have a big impact on its operational carbon footprint due to the available energy mix. Choosing the right location is challenging for both data center designers and operators. Designers need to decide where to build new data centers. Operators must decide where to execute submitted jobs when accessing multiple data centers. In both cases, insight into the effect of location on the operational carbon footprint is required.

***UC-Hardware*** *Selecting hardware upgrades:* A designer responsible for upgrading a data center hardware has to make choices within a limited budget. With a wide range of hardware options, deciding what to install can be difficult. To make informed decisions, designers must understand the impact of hardware changes.

## 4 FOOTPRINTER

We propose FootPrinter, an *energy-aware* discrete data center simulator based on the OpenDC framework. FootPrinter takes as input the hardware configuration of a data center and workload traces, and uses simulation to determine the energy footprint. The energy footprint is combined with the energy mix of the data center's region to determine the operational carbon footprint of the data center during the execution of the given workloads.

Figure 4 shows the architecture of FootPrinter and illustrates how it could be used by data center operators. Using the FootPrinter starts at the real data center ❶. Over time, different workloads ❶ are submitted to the servers ❷, and the operations software ❸ is used to decide when, where, and how these workloads are executed. The activity of the data center is monitored during operations and recorded. To use FootPrinter, three pieces of information are required as input data Ⅱ:

- ❹ *Workload traces* that describe when jobs are submitted and hardware requirements of each job. The trace also describes the computational demand over time. FootPrinter is designed to work with traces of any sample frequency. However, providing traces with higher frequency will result in more precise results.
- ❺ *Hardware and environment specifications* that describe the hardware used by the datacenter. To determine the carbon footprint, it is also important to define where a data center is located.
- ❻ *Operational techniques* that define how and when jobs are run. Important factors are the scheduling and resource allocation policies.

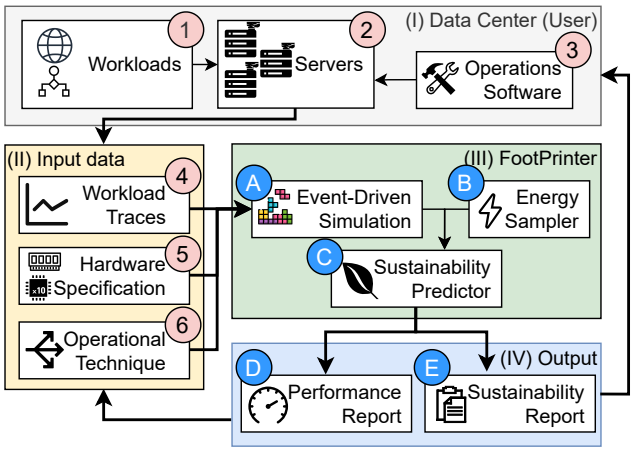



**Figure 4: A diagram of the FootPrinter functionality. Four areas are defined: The Data Center which is controlled by the user ❶, the input data gathered from the data center Ⅱ, The FootPrinter which simulated the input data Ⅲ, and the output Ⅳ.**

The input data is sent to the FootPrinter to replay. The FootPrinter architecture Ⅲ consist of the following components:

- Ⓐ The *Event-Driven Simulator* replays the given workload traces on the given data center configuration. During the run, the simulator is sampled for performance metrics and energy usage. The frequency of sampling can be chosen to best fit the current experiment. Higher frequency will result in more precision at a cost of increasing the simulation time.
- Ⓑ The *Energy Sampler* determines the carbon intensity of the grid while the simulation is run. Whenever the event-driven simulator is sampled, the carbon intensity of the grid is needed. The energy mix of the grid is sampled using the Python API[7] of the ENTSO-E Transparency Platform[8].
- Ⓒ The *Sustainability Predictor* aggregates the results of the simulation into sustainability metrics, such as the total carbon emitted and the carbon emission over time. These metrics can be used to determine the operational carbon footprint of the data center during the workload.

FootPrinter generates two types of output Ⅳ. First, the Performance Report Ⓓ shows the performance of the data center during the provided workload. Examples of performance metrics are the time of completion, or average CPU utilization. Next to the performance of the data center, a sustainability report Ⓔ is made. Examples of sustainability metrics are the energy usage, or the carbon emitted. Designing data centers is a difficult process, in which often improvements in sustainability are connected to decreases in performance. FootPrinter reports both sides to provide the data center operators with a complete insight.

[7] https://github.com/EnergieID/entsoe-py
[8] https://transparency.entsoe.eu/dashboard/show

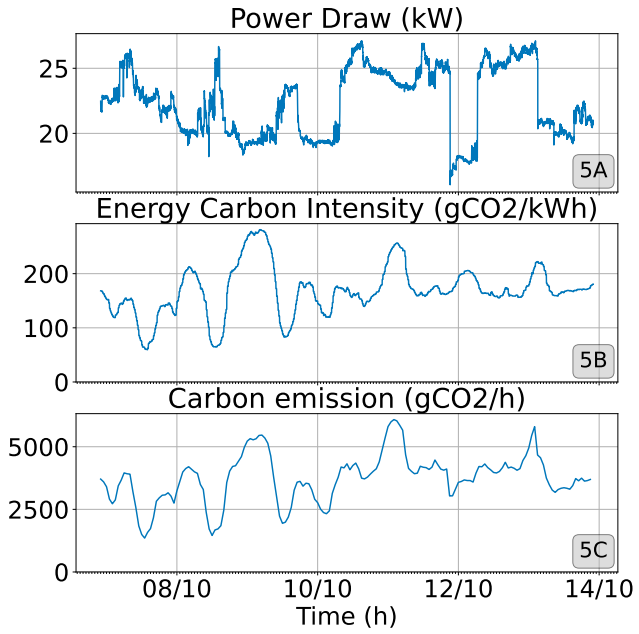



**Figure 5: The Carbon emission of a workload over time, determined using FootPrinter. Graph 5A shows the power draw over time. Graph 5B shows the carbon intensity of the grid during the workload. Graph 5C combines the two other graphs, showing the carbon emission during the workload.**

## 5 EXPERIMENTS

This section demonstrates how FootPrinter can be used in different use cases from section 3. The accuracy of FootPrinter is validated by comparing it to an empirically measured energy usage trace.

### 5.1 Operational Carbon Footprint

We use FootPrinter to determine the operational carbon footprint of a data center (*UC-Footprint*). To illustrate the process, we simulate a workload trace gathered from the SURF Lisa[9] cluster, an HPC data center in the Netherlands. The workload consists of 7,850 jobs executed over seven days. The duration of the jobs ranges from less than an hour to several days. The CPU demand is sampled at a 30-second interval for each job in the trace. The workload is run on a data center comprising 277 physical machines. FootPrinter replays this trace on a mid-range laptop (Intel Core I7-8750H Processor[10]) in 10 seconds. This allows for rapid experimentation mentioned in section 3.

Figure 5 depicts the process of determining operational carbon footprint using FootPrinter. Figure 5A shows the simulator-determined power draw of the data center during the workload, sampled every 30 seconds. The graph depicts the power draw of the entire data center. However, FootPrinter can also provide similar graphs for specific nodes or jobs. The aggregate power draw varies in the range of 16 to 28 kW. The energy usage at a sample can be determined by multiplying the power draw and the time since the previous sample. Figure 5B depicts the carbon intensity of the grid sampled from ENTSO-E. The difference in carbon intensity during the chosen period is significant, ranging between 100 and 400 gCO2/h. Figure 5C depicts the carbon emission during the workload. Carbon emission at a sample can be calculated by multiplying the energy usage at a sample with the carbon intensity. The carbon emission is primarily influenced by the carbon intensity, due to the much higher variability in the carbon intensity compared to the power draw. This demonstrates the importance of measuring the carbon footprint directly, instead of just energy usage.

[9]https://www.surf.nl/en/lisa-computing-cluster-extra-computing-power-for-research

[10]https://ark.intel.com/content/www/us/en/ark/products/134906/intel-core-i7-8750h-processor-9m-cache-up-to-4-10-ghz.html

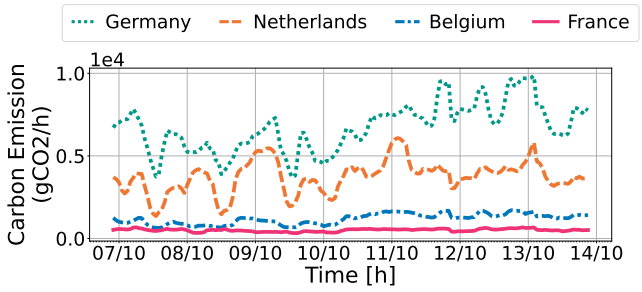



**Figure 6: The carbon emission during the same workload simulated executed on the same data center located in four different locations.**

### 5.2 Selecting location

FootPrinter can be used to compare the impact of building or expanding the data center infrastructure in multiple locations (*UC-Location*). Figure 6 depicts the effect of the data center location on its carbon emission. The workload introduced in subsection 5.1 is replayed on the same data center in different locations. France and Belgium perform much better than the Netherlands and Germany. This is because France and Belgium source around half of their energy from nuclear power plants emitting almost no carbon. The Netherlands and Germany, however, rely more on energy sources such as coal, which is very carbon intensive.

### 5.3 Validation

To quantify the accuracy of our simulator, we compare the power draw of a workload determined by the simulator, to the real-world power draw of the same workload. We use the same workload as used in subsection 5.1. Figure 7 shows the simulated power draw determined by FootPrinter and the real-world power draw. We determine the accuracy of the estimation using three different metrics. Each metric is calculated separately for all points, the points in which FootPrinter *underestimates* (underestimation error), and the points in which FootPrinter *overestimates* the power draw (overestimation error).

The first metric of estimation accuracy is the Mean Absolute Percentage Error (MAPE), a popular measure of the accuracy of forecasting methods. MAPE is commonly used to determine forecast accuracy because of its intuitive interpretation in terms of relative error [16]. MAPE is a relative error measure that uses absolute values to keep the positive and negative errors from canceling one another out [33] and is calculated using Equation 4:

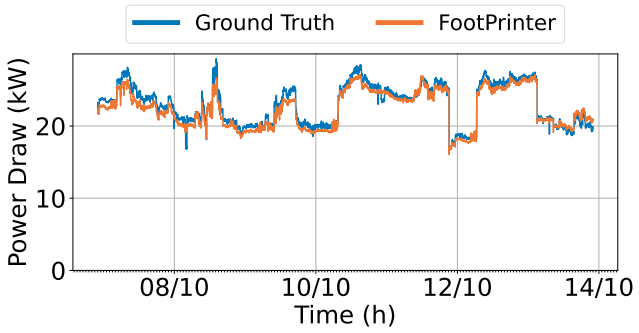



**Figure 7: The power draw of a data center during a given workload simulated by the FootPrinter tool compared to the actual Power Draw of the data center.**

$$MAPE[\%] = \frac{1}{n}\sum_{t=0}^{n} |\frac{P_t - P'_t}{P_t}| \times 100 \quad (4)$$

In which $P_t$ and $P'_t$ are the actual and simulated power draw at sample $t$ and $n$ is the number of samples. Comparing FootPrinter to the ground truth results in a MAPE total error of 3.15%, underestimation error of 3.19%, and overestimation error of 2.93%.

The second metric of prediction accuracy is the Normalized Absolute Differences (NAD). NAD describes the total error of the prediction divided by the sum of the ground truth and is calculated using Equation 5:

$$NAD[\%] = \frac{\sum_{t=0}^{n} |P_t - P'_t|}{\sum_{t=0}^{n} P_t} \times 100 \quad (5)$$

In which $P_t$ and $P'_t$ are the actual and simulated power draw at sample $t$ and $n$ is the number of samples. Comparing FootPrinter to the ground truth results in a NAD total error of 3.17%, underestimation error of 3.22%, and overestimation error of 2.83%.

Finally, we look at the distribution of the errors. Figure 8 shows the percentage of time points with an error less than the given threshold. Over half of the points have an error less than 3%, and 93% an error less than 6%.

## 6 RELATED WORK

The research community has built many high-quality simulators that provide a rich set of features to build upon [6, 8]. CloudSim [9] is the closest to OpenDC, the simulator used in this paper. CloudSim offers a number of single-feature simulators such as CloudAnalyst [39], iFogSim [21], and WorkflowSim [11]. However, the single focus of these simulators makes it challenging to combine without extensive engineering. In contrast, OpenDC is a flexible general purpose simulator that supports various different features. Building FootPrinter op OpenDC guarenties support for a varied applictions.

Extending simulators to estimate the carbon footprint of a data centers is not a novel idea. In their paper from 2022, Song et al. discuss over 100 papers working on data center carbon footprint in the last ten years, in which 75% used simulators in their experiment [37]. Most of the works discussed extend third party simulators to estimate carbon footprint. The Most popular simulator for this purpose is Cloudsim [15, 25, 42], but other simulators, such as SimGrid [5],

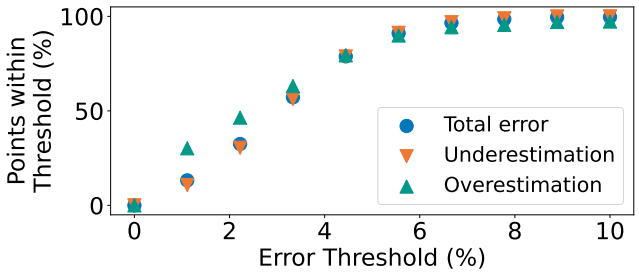



**Figure 8: The distribution of the error of samples. Each point represents the percentage of samples with an error less than the specified threshold.**

EcoMultiCloud [2], and iFogSim [41], are also used. Because of the single-feature nature of the simulators used, most of these tools are very specialized for their specific purpose. In contrast, FootPrinter is more general purpose. Another distinction is that many tools focus on single green energy sources, such as solar [26, 27], or wind [17]. FootPrinter is not dependent on any specific type of energy source.

## 7 CONCLUSION

This work introduces FootPrinter, a first-of-its-kind tool that uses simulation to determine the operational carbon footprint of a data center. FootPrinter replays workload traces to determine the energy usage and carbon emission during the workload execution. FootPrinter is designed to work with any trace granularity to make it accessible to all data center operators. We have validated FootPrinter by comparing the simulated energy usage to the real-world energy usage. FootPrinter can simulate energy usage with a Mean Average Percentage Error of less than 3.15%.

We discussed three use cases highlighting challenges for data center designers and operators who want to evaluate the sustainability impact of their actions. In this paper, we showed how FootPrinter can be used to determine operational carbon footprint and compare data center locations. FootPrinter is an open-source tool and can be extended to support more use cases and provide more insights. We are already actively working on supporting hardware upgrades and their impact on performance and carbon footprint. Additionally, we are working on adding support for more elements that can influence the energy usage of a data center, such as temperature and humidity. Finally, while FootPrinter currently quantifies the operational carbon emissions of a data center, we believe it can be easily extended to also incorporate embodied carbon emissions.

## ACKNOWLEDGMENTS

This work is supported by EU Horizon Graph Massivizer (g.a. 101093202) and EU MSCA CloudStars projects (g.a. 101086248). This research is partly supported by a National Growth Fund through the Dutch 6G flagship project "Future Network Services". We wish to thank SURF for the valuable support.

## REFERENCES

[1] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Udit Gupta, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. 2023. Carbon Explorer: A

Holistic Framework for Designing Carbon Aware Datacenters. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 118–132. https://doi.org/10.1145/3575693.3575754

[2] Abada Ahmed and Marc St-Hilaire. 2018. *Renewable Energy Curtailment via Incentivized Inter-datacenter Workload Migration.* 143–157. https://doi.org/10.1007/978-3-319-94295-7_10

[3] Anders S. G. Andrae and Tomas Edler. 2015. On Global Electricity Usage of Communication Technology: Trends to 2030. *Challenges* 6, 1 (2015), 117–157. https://doi.org/10.3390/challe6010117

[4] Georgios Andreadis, Fabian Mastenbroek, Vincent van Beek, and Alexandru Iosup. 2021. Capelin: Data-Driven Capacity Procurement for Cloud Datacenters using Portfolios of Scenarios – Extended Technical Report. arXiv:2103.02060 [cs.DC]

[5] Gagangeet Singh Aujla and Neeraj Kumar. 2018. SDN-based energy management scheme for sustainability of data centers: An analysis on renewable energy sources and electric vehicles participation. *J. Parallel Distrib. Comput.* 117, C (jul 2018), 228–245. https://doi.org/10.1016/j.jpdc.2017.07.002

[6] Ilyas Bambrik. 2020. A Survey on Cloud Computing Simulation and Modeling. *SN Computer Science* 1, 5 (2020), 249.

[7] Noman Bashir, David Irwin, Prashant Shenoy, and Abel Souza. 2022. Sustainable Computing – Without the Hot Air. arXiv:2207.00081 [cs.CY]

[8] James Byrne, Sergej Svorobej, Konstantinos M. Giannoutakis, Dimitrios Tzovaras, Peter J. Byrne, Per-Olov Östberg, Anna Gourinovitch, and Theo Lynn. 2017. A Review of Cloud Computing Simulation Platforms and Related Environments. In *CLOSER 2017 - Proceedings of the 7th International Conference on Cloud Computing and Services Science, Porto, Portugal, April 24-26, 2017*, Donald Ferguson, Víctor Méndez Muñoz, Jorge S. Cardoso, Markus Helfert, and Claus Pahl (Eds.). SciTePress, 651–663.

[9] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience* 41, 1 (2011), 23–50.

[10] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. 2014. Versatile, scalable, and accurate simulation of distributed applications and platforms. *J. Parallel and Distrib. Comput.* 74, 10 (2014), 2899–2917. https://doi.org/10.1016/j.jpdc.2014.06.008

[11] Weiwei Chen and Ewa Deelman. 2012. WorkflowSim: A toolkit for simulating scientific workflows in distributed environments. In *8th IEEE International Conference on E-Science, e-Science 2012, Chicago, IL, USA, October 8-12, 2012*. IEEE Computer Society, 1–8.

[12] European Commission. 2020. Stepping up Europe's 2030 climate ambition Investing in a climate-neutral future for the benefit of our people. https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:52020DC0562

[13] ACM Technology Council. 2021. Computing and climate change. https://doi.org/doi/pdf/10.1145/3483410

[14] J. Davis, D. Bizo, A. Lawrence, O. Rogers, M. Smolaks, L. Simon, and D. Donnellan. 2022. Uptime Institute Global Data Center Survey 2022. Uptime Institute.

[15] Inès De Courchelle, Tom Guérout, Georges Da Costa, Thierry Monteil, and Yann Labit. 2019. Green energy efficient scheduling management. *Simulation Modelling Practice and Theory* 93 (2019), 208–232. https://doi.org/10.1016/j.simpat.2018.09.011 Modeling and Simulation of Cloud Computing and Big Data.

[16] Arnaud de Myttenaere, Boris Golden, Bénédicte Le Grand, and Fabrice Rossi. 2016. Mean Absolute Percentage Error for regression models. *Neurocomputing* 192 (2016), 38–48. https://doi.org/10.1016/j.neucom.2015.12.114 Advances in artificial neural networks, machine learning and computational intelligence.

[17] Xiaowen Dong, Taisir El-Gorashi, and Jaafar Elmirghani. 2011. Green IP over WDM Networks: Solar and Wind Renewable Sources and Data Centres. 1–6. https://doi.org/10.1109/GLOCOM.2011.6134175

[18] Mariam Elgamal, Doug Carmean, Elnaz Ansari, Okay Zed, Ramesh Peri, Srilatha Manne, Udit Gupta, Gu-Yeon Wei, David Brooks, Gage Hills, and Carole-Jean Wu. 2023. Carbon-Efficient Design Optimization for Computing Systems. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems* (Boston, MA, USA) *(HotCarbon '23)*. Association for Computing Machinery, New York, NY, USA, Article 16, 7 pages. https://doi.org/10.1145/3604930.3605712

[19] Anshul Gandhi, Dongyoon Lee, Zhenhua Liu, Shuai Mu, Erez Zadok, Kanad Ghose, Kartik Gopalan, Yu David Liu, Syed Rafiul Hussain, and Patrick Mcdaniel. 2023. Metrics for Sustainability in Data Centers. *SIGENERGY Energy Inform. Rev.* 3, 3 (oct 2023), 40–46. https://doi.org/10.1145/3630614.3630622

[20] James Glanz. 2012. *Power, Pollution and the Internet.* https://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html

[21] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. 2017. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience* 47, 9 (2017), 1275–1296.

[22] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2022. ACT: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 784–799. https://doi.org/10.1145/3470496.3527408

[23] IEA. 2017. *Digitalisation and Energy.* https://www.iea.org/reports/digitalisation-and-energy

[24] IEA. 2022. *Data Centres and Data Transmission Networks.*

[25] Chao Li, Rui Wang, Depei Qian, and Tao Li. 2016. Managing Server Clusters on Renewable Energy Mix. *ACM Trans. Auton. Adapt. Syst.* 11, 1, Article 1 (feb 2016), 24 pages. https://doi.org/10.1145/2845085

[26] Chao Li, Wangyuan Zhang, Chang-Burm Cho, and Tao Li. 2011. SolarCore: Solar energy driven multi-core architecture power management. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. 205–216. https://doi.org/10.1109/HPCA.2011.5749729

[27] Longjun Liu, Hongbin Sun, Yang Hu, Jingmin Xin, Nanning Zheng, and Tao Li. 2015. Leveraging distributed UPS energy for managing solar energy powered data centers. (02 2015). https://doi.org/10.1109/IGCC.2014.7039150

[28] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. 2020. Big Tech Gets Caught Up in Europe's Energy Politics. https://www-science-org.vu-nl.idm.oclc.org/doi/10.1126/science.aba3758

[29] Fabian Mastenbroek, Georgios Andreadis, Soufiane Jounaid, Wenchen Lai, Jacob Burley, Jaro Bosch, Erwin van Eyk, Laurens Versluis, Vincent van Beek, and Alexandru Iosup. 2021. OpenDC 2.0: Convenient Modeling and Simulation of Emerging Technologies in Cloud Datacenters. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 455–464. https://doi.org/10.1109/CCGrid51090.2021.00055

[30] Rich Miller. 2021. The Sustainability Imperative: Green Data Centers and Our Cloudy Future. https://www.datacenterfrontier.com/special-reports/article/11428454/the-sustainability-imperative-green-data-centers-and-our-cloudy-future

[31] United Nations. 2015. *The Paris Agreement.* https://www.un.org/en/climatechange/paris-agreement

[32] Alberto Núñez, Jose Vázquez-Poletti, Agustín Caminero, Gabriel Castañé, Jesus Carretero, and Ignacio Llorente. 2012. ICanCloud: A Flexible and Scalable Cloud Infrastructure Simulator. *Journal of Grid Computing* 10 (03 2012), 185–209. https://doi.org/10.1007/s10723-012-9208-5

[33] Oracle. 2014. *Working with Planning: MAPE.* https://docs.oracle.com/en/cloud/saas/planning-budgeting-cloud/pfusu/insights_metrics_MAPE.html#GUID-C33B0F01-83E9-468B-B96C-413A12882334

[34] Interfovernmental panel on climate change. 2022. *IPCC PRESS RELEASE.* https://www.ipcc.ch/report/ar6/wg2/downloads/press/IPCC_AR6_WGII_PressRelease-English.pdf

[35] V. Reddy, B. Setz, G. K. Rao, G. Gangadharan, and M. Aiello. 2017. Metrics for Sustainable Data Centers. *IEEE Transactions on Sustainable Computing* 2, 03 (jul 2017), 290–303. https://doi.org/10.1109/TSUSC.2017.2701883

[36] April Roach and Ewa Krukowska. 2022. Recalibrating global data center energy-use estimates. https://www.bnnbloomberg.ca/big-tech-gets-caught-up-in-europe-s-energy-politics-1.1782670

[37] Jie Song, Peimeng Zhu, Yanfeng Zhang, and Ge Yu. 2022. Versatility or validity: A comprehensive review on simulation of Datacenters powered by Renewable Energy mix. *Future Generation Computer Systems* 136 (11 2022), 326–341. https://doi.org/10.1016/j.future.2022.06.008

[38] Petroc Taylor. 2023. *Data center average annual power usage effectiveness (PUE) worldwide 2007-2023.* https://www.statista.com/statistics/1229367/data-center-average-annual-pue-worldwide/

[39] Bhathiya Wickremasinghe, Rodrigo N. Calheiros, and Rajkumar Buyya. 2010. CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications. In *24th IEEE International Conference on Advanced Information Networking and Applications, AINA 2010, Perth, Australia, 20-13 April 2010*. IEEE Computer Society, 446–452.

[40] Jackson Woodruff, David Schall, Michael F.P. O'Boyle, and Christopher Woodruff. 2023. When Does Saving Power Save the Planet?. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems* (Boston, MA, USA) *(HotCarbon '23)*. Association for Computing Machinery, New York, NY, USA, Article 20, 6 pages. https://doi.org/10.1145/3604930.3605719

[41] Chenhan Xu, Kun Wang, Peng Li, Rui Xia, Song Guo, and Minyi Guo. 2020. Renewable Energy-Aware Big Data Analytics in Geo-Distributed Data Centers with Reinforcement Learning. *IEEE Transactions on Network Science and Engineering* 7, 1 (2020), 205–215. https://doi.org/10.1109/TNSE.2018.2813333

[42] Minxian Xu and Rajkumar Buyya. 2020. Managing renewable energy and carbon footprint in multi-cloud computing environments. *J. Parallel and Distrib. Comput.* 135 (2020), 191–202. https://doi.org/10.1016/j.jpdc.2019.09.015

[43] Runlin Zhou, Yingjie Shi, and Chunge Zhu. 2013. AxPUE: Application level metrics for power usage effectiveness in data centers. In *2013 IEEE International Conference on Big Data*. 110–117. https://doi.org/10.1109/BigData.2013.6691705

# Peeking Behind the Serverless Implementations and Deployments of the Montage Workflow

Simon Triendl
csav3166@student.uibk.ac.at
University of Innsbruck
Department of Computer Science
Innsbruck, Tyrol, Austria

Sashko Ristov
sashko.ristov@uibk.ac.at
University of Innsbruck
Department of Computer Science
Innsbruck, Tyrol, Austria

## ABSTRACT

The development of serverless scientific workflows is a complex and tedious procedure and opens several challenges in how to compose workflow processing steps as serverless functions and how much memory to assign to each serverless function, which affects not only the computing resources, but also the networking communication to the cloud storage. Merging multiple processing steps into a single serverless function (fusion) reduces the number of invocations, but restricts the developer to assign the maximum required memory of all fused processing steps, which may increase the overall costs.

In this paper, we address the aforementioned challenges for the widely used Montage workflow. We created three different *workflow implementations* (*fine*, *medium*, and *coarse*) for two cloud providers AWS and GCP and deployed workflow functions with different memory assignments 135 MB, 512 MB, and 1 GB (*function deployments*). Our experiments show that many Montage functions run cheaper and faster with more memory on both providers. Consequently, selecting the most cost-effective memory configuration, as opposed to the minimal memory, resulted in a reduction of the makespan by 67.27 % on AWS and 10.93 % on GCP. Applying the same to workflow implementations with fewer functions (coarse) led to a further reduction in the makespan by 24.98 % on AWS and 12.96 % on GCP, while simultaneously reducing the total cost by 5.33 % and 1.99 %, respectively. Surprisingly, the fastest implementation was the medium implementation executed on AWS.

## CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

## KEYWORDS

cost, FaaS, performance, serverless, workflows.

**ACM Reference Format:**
Simon Triendl and Sashko Ristov. 2024. Peeking Behind the Serverless Implementations and Deployments of the Montage Workflow. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3651420

This work is licensed under a Creative Commons Attribution-NonCommercial International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651420

## 1 INTRODUCTION

Serverless computing is a scalable and cost-effective execution environment for *data-intensive applications*, such as scientific workflow applications, whose complexity and scalability grow [18, 23]. Domain experts code processing steps as serverless functions and orchestrate them into *serverless workflows* [2, 7, 9, 21, 31, 38, 45, 50]. Such workflows include complex applications from astrophysics (e.g., Montage [5]), bioinformatics (e.g., BWA [27]), earthquake simulations (e.g., Cybershake [29]), and many more.

Although the heterogeneous nature of federated clouds [3, 28, 34, 40, 44, 49] brings many benefits in terms of cost [14], execution time [34, 38, 43], and scalability [22, 41], it raises several challenges. First, developers may map the workflow processing steps in a *fine-grained* manner, where each method of the workflow tasks may be deployed as a separate serverless function or a *coarse-grained* approach, in which multiple processing steps are merged in a separate serverless function. While the former approach offers the highest level of granularity and usually minimizes the cost, the latter usually improves the performance as it minimizes data transfers and uses more memory, which often leads to higher costs. We refer to each of these mappings as a *workflow implementation*. Once the functions of the workflow are coded, they are deployed on the selected provider and assigned with a specific amount of memory, often known as *function deployment*.

In this paper, we peek behind the different Montage workflow implementations and various function deployments to investigate how they affect the overall cost and performance. We used two cloud providers AWS and GCP and derived interesting conclusions. The evaluation showed that many workflow functions benefit both in terms of cost and performance when assigned with more memory. Moreover, the coarse implementations additionally reduce the overall cost while improving performance. Surprisingly, the medium implementation of Montage achieved the fastest makespan on AWS. While the workflow community [12] recommends that a workflow orchestrator should make intelligent decisions about the placement of workflow tasks across different sites in the continuum, our results show that the workflow orchestrator should consider fusion and fission of the workflow tasks, as well as the assigned memory to each newly created task.

The remaining part of the paper is organized in five sections. We first present details for the Montage workflow and its processing steps in Section 2. Further on, Section 3 presents various implementation and deployment challenges and how they affect performance and cost. In Section 4, we evaluate several deployments of the Montage workflow functions and determine the cheapest and fastest implementations on two providers AWS and GCP. In Section 5, we

position our work compared to the related work and in Section 6, we conclude the paper and present our plans for future work.

## 2 INTRODUCTION TO THE MONTAGE WORKFLOW

Montage [5], created by the NASA/IPAC Infrared Science Archive, is an open-source toolkit to assemble astronomical images into custom mosaics. It consists of multiple independent modules indended to be used in a choreography that exploits the parallelization of several processing steps. This orchestration of modules is commonly referred to as *Montage workflow*. The Montage workflow is widely used by the scientist due to its high computation and communication complexity [1, 2, 13, 23, 31–33].

At a high level, the processing steps to compute a mosaic involve initially calculating the geometry of the mosaic, followed by re-projecting the input images, stored in Flexible Image Transport System (FITS) format. Subsequently, background radiation correction is applied to ensure uniformity across the mosaic. Finally, the re-projected and background-corrected images are co-added to generate the mosaic.

The Montage toolkit implements all of these computations with a higher granularity. Additionally, certain stages necessitate the dynamic generation of input variables for subsequent stages. Therefore, a naive fine-grained implementation comprises 19 stages.

(0)-(1) prepare `mProjectPP I-II`: Prepares the parameters for the `mProjectPP` instances.
(2) `mProjectPP`: Performs parallel fast re-projection of the input FITS files according to the region header file.
(3)-(6) prepare `mDiffFit I-IV`: Extracts the header information from the input FITS files and computes a list of overlapping images. Based on the overlaps, the parameters for the `mDiffFit` and `mFitPlane` instances are created.
(7) `mDiff`: Performs parallel image difference between a pair of re-projected images and creates a new FITS file.
(8) `mFitPlane`: Parallelly fits a plane to the FITS files generated by `mDiff`.
(9) prepare `mConcatFit`: Prepares the parameters for `mConcatFit`.
(10) `mConcatFit`: Merges the multiple plane fit parameter into a single file for `mBgModel`.
(11) `mBgModel`: Creates a background radiation model to determine a set of corrections stored in a table.
(12)-(13) prepare `mBackground I-II`: Prepares the corrections table and the parameters for the `mBackground` instances.
(14) `mBackground`: Performs parallel corrections on the re-projected input FITS files.
(15) `mImgTbl`: Extracts the header information from the corrected FITS files and stores them in a table.
(16) `mAdd`: Co-add the corrected FITS files according to the header region file.
(17) `mShrink`: Reduces the size of the co-added FITS file.
(18) `mViewer`: Converts the shrunk FITS file to a PNG or JPEG format.

## 3 IMPLEMENTATION AND DEPLOYMENT CHALLENGES

In this section, we discuss the challenges that developers face when they decide which implementation of a workflow to use and which function deployments of that implementation.

### 3.1 Deployment challenges

Cloud providers offer fine-grained configuration of RAM memory to their serverless functions. Developers can deploy their functions starting from 128 MB up to tens of gigabytes, depending on the provider. However, providers use different approaches for the CPU. AWS claims that the CPU scales linearly as the assigned memory. Azure configures the memory dynamically based on the need, while GCP users can configure the CPU, as well, but are additionally charged for the CPU resources.

Function deployments with more memory usually achieve better performance [24] based on the Gustafson's Law [16]. Moreover, function deployments on AWS with more than 1.5 GB achieve the maximum bandwidth of the underlying virtual machines [47], thereby transferring data to the collocated storage faster than the function deployments with lower memory [26, 48]. Still, the speedup when increasing the resources are limited to the linear speedup, often reaching the sublinear speedup.

### 3.2 Implementation challenges

The cost and performance of a workflow is affected by its implementations. We showed earlier that Montage comprises 19 processing steps, some of which are nested in a parallel loop. If all processing steps are merged in a single function, then the invocation overhead will be minimized as the workflow calls a single function [35], but in that case, the developers lose parallelism. Therefore, fusion of the processing steps should be considered mainly for a sequence of sequential processing steps. Unfortunately, providers limit the size of the function codes, which additionally restricts the decision how many processing steps to merge and deploy in a single function. Another challenge in the fusion of multiple processing steps is the amount of memory that should be assigned to the equivalent function. For instance, if a processing step $ps_1$ needs 128 MB and the subsequent processing step $ps_2$ requires 2 GB, then the fused function $f_{12}$ also needs 2 GB, which additionally may increase the costs since the processing step $ps_1$ is assigned with more memory than required.

## 4 EXPERIMENTAL DESIGN

In this section, we introduce our implementation and assess how different function deployments and workflow implementations influence the cost and performance of the Montage workflow on the two public cloud providers AWS and GCP. Our implementation is written in Java, utilizing JDK version 17. We encapsulated each Montage toolkit executable within a method, offering a high-level interface for the parameters. Upon method invocation, the executable is executed in a process. We utilized Montage version 6.0 for this study.

We devised three workflow implementations, denoted as the *fine*, *medium*, and *coarse* implementations, as presented in Fig. 1.

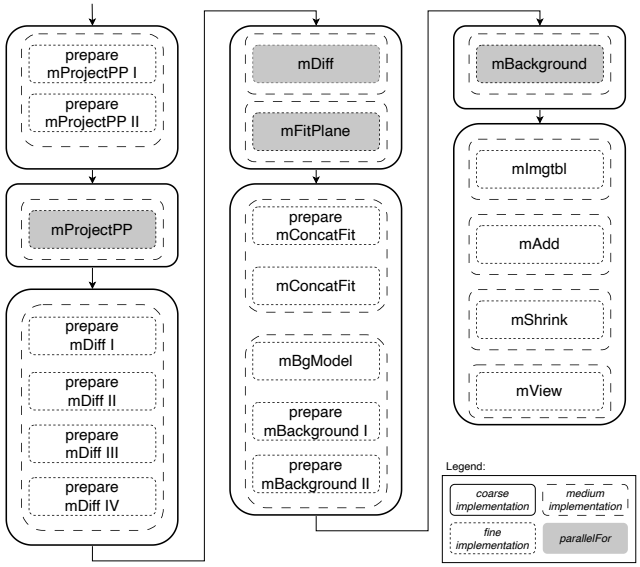



**Figure 1: Composition of *coarse*, *medium* and *fine* Montage workflow implementations, depicted with solid, dashed, and dotted rectangles, respectively.**

The fine implementation (dotted boxes in Fig. 1) focuses on dividing the Montage computations with the highest granularity, achieved by segregating each method invocation into individual functions. The medium implementation (dashed boxes in Fig. 1) reflects the employed workflow[17] that can be executed with the xAFCL serverless workflow management system [38]. Lastly, the coarse implementation (solid boxes in Fig. 1) targets a minimal granularity approach, also used by Deelman [13] and Berriman [5]. All three implementations and their compositions run the same work and provide the same output, but their computations are grouped differently in serverless functions, leading to 19, 12, and 7 functions for the fine, medium, and coarse implementations, respectively. Note that the functions marked with gray boxes are executed multiple times using data parallelism. Also, there are other implementations of Montage that use 9 functions [31, 32], which we did not consider since they are very similar to the coarse implementation.

## 4.1 Experiment setup

We deployed the functions of all three workflow implementations on GCP as 2nd generation functions in the `europe-west1` region, and on AWS in the `eu-central-1` region, as the closest regions to University of Innsbruck to minimize the invocation overhead [35]. Each function was deployed with 135 MB, 512 MB, and 1 GB of memory. The usage of 135 MB was necessary on GCP due to its minimal memory assignment being 135 MiB, roughly equivalent to 135 MB. Each function was configured with the maximum timeout duration to prevent premature failure. The storage, which were respectively hosted on Amazon S3 and Google Cloud Storage, were collocated in the same region for both platforms, based on the recent reports to collocate the functions closer to the data [42, 43].

To conduct these experiments we used a custom workflow execution engine written in Python. Parallelization was achieved using a thread pool, with the number of workers regulating the concurrency of the execution. Due to account limitations on both providers, we set the concurrency level to 10 and the block size to 1. This constraint does not affect the results, as the measured execution time reflects the function's internal wall clock time. Thus, factors such as invocation latency and cold starts do not influence the measurements.

Each workflow implementation, along with its corresponding functions deployments, underwent 5 executions like in other recent works [10, 17, 35], and the median execution time of each function was considered. For functions invoked in parallel, the median execution time of all parallel instances was calculated and then extrapolated to determine the total runtime. We choose the median over the mean to mitigate the impact of outliers.

The runtime cost is calculated based on publicly available pricing information from AWS[1] and GCP[2]. Because the storage is collocated, file transfer expenses are confined to the number of file accesses (downloads and uploads). Both AWS[3] and GCP[4] adhere to the same pricing model in this regard.

## 4.2 Memory impact

We compiled the runtime and its associated cost, excluding file transfer expenses, for each function and function deployment within the fine workflow implementation into a table. Our objective is to identify deployments that minimize runtime costs. To make the detection easier, we computed the cost for 1 million invocations and highlighted the cheapest deployments in bold. Dash symbols indicate that the deployment's memory was insufficient to execute the function.

*4.2.1 AWS.* In Table 1 we depicted the results for AWS. Observing the runtimes of the functions reveals that deployments with higher memory result in a faster execution. While the cost scales linearly with both runtime and memory assignment, some functions experience a super-linear speedup due to increased memory assignment [39]. Consequently, deployments with higher memory become both more cost-effective and faster compared to their lower-spec counterparts. Functions 2, 7-8, 11, and 14 highlight that this phenomenon is not universal but rather dependent on the behavior of each specific function. Notable are the functions 0, 1, 6, 12-13, 15, and 17, whose deployments with even 1 GB dominate the other deployments with less memory, both in terms of runtime and cost. Function 6 experiences the highest impact running at just 0.77 % of the time and accounting for 5.76 % of the cost compared to its deployment with 135 MB. When comparing the cost-optimal function deployments to deployments with minimal feasible memory, we observe a reduction in makespan from 434.96 s to 142.33 s, representing a decrease of 67.27 %. Simultaneously, the cost decreased from $11.57 to $11.19, a reduction of 3.28 %.

[1] https://aws.amazon.com/lambda/pricing/
[2] https://cloud.google.com/run/pricing
[3] https://aws.amazon.com/s3/pricing/
[4] https://cloud.google.com/storage/pricing

**Table 1: Runtime and cost for different deployments of the *fine* workflow implementation on AWS.**

| $f$ | *Runtime (s)* | | | *Cost ($) for 1M invocations* | | |
|---|---|---|---|---|---|---|
| | *135MB* | *512MB* | *1024MB* | *135MB* | *512MB* | *1024MB* |
| 0 | 0.285 | 0.055 | **0.005** | 0.63 | 0.45 | **0.09** |
| 1 | 0.378 | 0.082 | **0.005** | 0.837 | 0.68 | **0.08** |
| 2 | 635.943 | **165.571** | 84.685 | 1409.00 | **1382.79** | 1412.97 |
| 3 | - | 18.054 | **7.152** | - | 150.78 | **119.33** |
| 4 | - | 7.559 | **2.187** | - | 63.13 | **36.49** |
| 5 | - | 4.859 | **0.921** | - | 40.58 | **15.36** |
| 6 | 11.462 | 2.830 | **0.088** | 25.39 | 23.63 | **1.46** |
| 7 | **2409.051** | 653.139 | 334.195 | **5337.54** | 5454.82 | 5576.06 |
| 8 | 613.759 | **159.886** | 80.283 | 1359.85 | **1335.32** | 1339.52 |
| 9 | - | 4.440 | **0.672** | - | 37.07 | **11.20** |
| 10 | - | 11.178 | **5.328** | - | 93.35 | **88.89** |
| 11 | **17.563** | 9.893 | 3.209 | **38.91** | 82.62 | 53.543 |
| 12 | 12.978 | 3.133 | **0.124** | 28.75 | 26.16 | **2.06** |
| 13 | 0.514 | 0.115 | **0.005** | 1.14 | 0.96 | **0.08** |
| 14 | - | **202.668** | 101.904 | - | **1692.61** | 1700.26 |
| 15 | 116.177 | 30.065 | **13.166** | 257.40 | 251.09 | **219.66** |
| 16 | - | 84.068 | **38.166** | - | 702.11 | **636.79** |
| 17 | 87.483 | 22.823 | **9.570** | 193.82 | 190.60 | **159.67** |
| 18 | - | 17.204 | **6.994** | - | 143.68 | **116.69** |

**Table 2: Runtime and cost of different deployments of the *fine* workflow implementation on GCP.**

| $f$ | *Runtime (s)* | | | *Cost ($) for 1M invocations* | | |
|---|---|---|---|---|---|---|
| | *135MB* | *512MB* | *1024MB* | *135MB* | *512MB* | *1024MB* |
| 0 | **0.04** | 0.005 | 0.006 | **0.43** | 0.91 | 1.63 |
| 1 | **0.085** | 0.007 | 0.005 | **0.43** | 0.91 | 1.63 |
| 2 | - | 210.939 | **115.112** | - | 1956.21 | **1916.01** |
| 3 | - | **8.151** | 5.672 | - | **75.31** | 93.34 |
| 4 | - | **4.568** | 3.57 | - | **42.24** | 58.95 |
| 5 | - | **1.878** | 1.324 | - | **17.45** | 22.92 |
| 6 | 1.302 | **0.190** | 0.235 | 6.05 | **1.83** | 4.91 |
| 7 | - | **247.796** | 158.587 | - | **2330.92** | 2770.85 |
| 8 | - | **84.89** | 54.162 | - | **906.47** | 923.61 |
| 9 | **1.886** | 1.683 | 1.059 | **8.21** | 15.61 | 18.01 |
| 10 | - | **25.667** | 18.254 | - | **236.03** | 299.68 |
| 11 | - | **4.516** | 2.811 | - | **42.24** | 47.49 |
| 12 | 1.384 | **0.361** | 0.304 | 6.05 | **3.67** | 6.55 |
| 13 | **0.03** | 0.005 | 0.005 | **0.43** | 0.91 | 1.63 |
| 14 | - | **214.309** | 126.167 | - | **1983.76** | 2112.52 |
| 15 | - | **8.863** | 6.334 | - | **81.73** | 104.80 |
| 16 | - | - | **38.58** | - | - | **632.12** |
| 17 | - | 23.909 | **13.083** | - | 220.41 | **214.52** |
| 18 | - | **10.436** | 6.294 | - | **96.43** | 103.17 |

*4.2.2 GCP.* Table 2 presents the results for GCP, which are comparable to, but not as pronounced as for AWS. Only functions 2, 12 and 17 demonstrate cheaper execution with higher memory deployments. Note that the functions 12 and 17 do not experience this effect on AWS. It is noteworthy that GCP rounds the execution time to the nearest 100 ms increment, which mitigates the speed-up effect for short-running functions such as 0, 1, and 13. Furthermore, GCP appears to be more restrictive with memory, as evidenced by several functions that have sufficient memory on AWS but fail on GCP. This is particularly apparent for function 16, which requires a minimum of 1 GB to execute. One possible reason for this discrepancy may be that the ephemeral storage, used by the Montage executables and their input files, scales with assigned memory on GCP. In contrast on AWS, it is set by default to 512 MB, which provides sufficient storage for most functions. While we still observe a reduction in makespan comparing the cost-optimal deployments with the minimal feasible memory deployments from 147.82 s to 131.67 s, equivalent to a 10.93 % reduction, the improvement is considerably smaller compared to AWS. When we apply the same to the runtime cost, we observe a reduction from $8.64 to $8.59, yielding a negligible 0.05 % decrease.

## 4.3 Fine vs. Medium vs. Coarse

We further applied the same approach as described in Section 4.2 to the medium and coarse implementations of the Montage workflow.

*4.3.1 Cost-optimal memory configuration.* The optimal function deployments for cost are presented in Table 3, which indicates that the optimal memory configuration for fused functions is not necessarily equivalent to that of separated functions. Two such examples can be seen when comparing AWS functions 7 and 8 in the medium and coarse implementations. While the separated functions worked most cost-effectively with 512 MB, the merged function is optimal with 135 MB. Conversely, for GCP, the exact same functions work optimally at 512 MB when separated, but when fused, they are optimal at 1 GB. Another notable example is AWS function 7 in both the fine and medium implementations. Despite their equivalence, the optimal deployment shifts from 135 MB in the fine implementation to 512 MB in the medium implementation. This discrepancy may be attributed to variations in the performance of the cloud provider. A similar effect is observed with GCP functions 15 and 18 in the fine and medium implementations. Increasing the number of iterations may help mitigate such results in the future.

*4.3.2 Makespan analysis.* To assess the overall impact of the cost-effective function deployments on the workflow implementations, we computed the makespan of each workflow along with its associated cost. Assuming no account-specific concurrency limit, we determined the makespan with maximum concurrency for each parallel function (2, 7, 8, and 14). Consequently, the makespan represents the total sum of all non-parallel function runtimes and the maximum runtime of any parallel instances. The different workflow implementations not only affect the optimal function deployments but also influence the number of files that need to be transferred. Fusing multiple sequential functions into one reduces file transfers, thereby affecting the runtime and cost. Hence, we computed costs both with and without factoring in file transfers. The makespan results are visually presented in Figure 2, while the cost is illustrated in Figure 3. When analyzing the makespan on AWS, the medium implementation proves to be the fastest, completing in 106.77 s, which is 93.08 % of the time taken by the coarse implementation (114.70 s), and 75.02 % of the time taken by the fine implementation (143.33 s). This result is supported by Table 3, where it is evident that the medium implementation utilizes a function deployment with either equal or higher memory assignment compared to the

**Table 3: Cost optimal memory configurations in MB.**

| $f$ | Instances | AWS | | | GCP | | |
|---|---|---|---|---|---|---|---|
| | | *fine* | *medium* | *coarse* | *fine* | *medium* | *coarse* |
| 0 | 1 | 1024 | 1024 | 512 | 135 | 135 | 135 |
| 1 | 1 | 1024 | | | 135 | | |
| 2 | 30 | 512 | 512 | 512 | 1024 | 512 | 512 |
| 3 | 1 | 1024 | 1024 | 512 | 512 | 512 | 512 |
| 4 | 1 | 1024 | | | 512 | | |
| 5 | 1 | 1024 | | | 512 | | |
| 6 | 1 | 1024 | | | 512 | | |
| 7 | 141 | 135 | 512 | 135 | 512 | 512 | 1024 |
| 8 | 141 | 512 | 512 | | 512 | 512 | |
| 9 | 1 | 1024 | 1024 | 512 | 135 | 512 | 512 |
| 10 | 1 | 1024 | | | 512 | | |
| 11 | 1 | 135 | 1024 | | 512 | 1024 | |
| 12 | 1 | 1024 | | | 512 | | |
| 13 | 1 | 1024 | | | 135 | | |
| 14 | 30 | 512 | 512 | 512 | 512 | 512 | 512 |
| 15 | 1 | 1024 | 1024 | 1024 | 512 | 1024 | 1024 |
| 16 | 1 | 1024 | 1024 | | 1024 | 1024 | |
| 17 | 1 | 1024 | 1024 | | 1024 | 1024 | |
| 18 | 1 | 1024 | 1024 | | 512 | 1024 | |

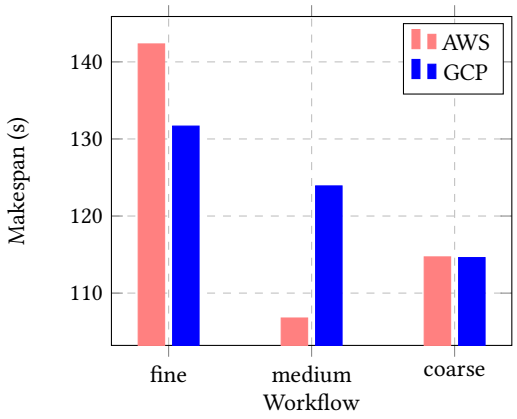



**Figure 2: Makespan of cost-optimal Montage implementations on AWS and GCP.**

other implementations. The reason for the coarse implementation being the second fastest may be attributed to the reduced number of file transfers, coupled with the utilization of function deployments with either equal or less memory compared to the medium implementation. The longer makespan in the fine implementation can be explained by examining function 7, which has a deployment of 135 MB and its associated runtime in Table 1. Although executed in parallel with 141 instances, its presence significantly impacts the critical path, thus extending the makespan.

Contrary to AWS, on GCP, the coarse implementation executes the fastest, completing in 114.61 s, which is 92.49 % of the time taken by the medium implementation (123.91 s) and 87.04 % of the time taken by the fine implementation (131.67 s). While there are minimal changes in the most cost-effective deployments between the coarse and medium implementations, as depicted in Table 3,

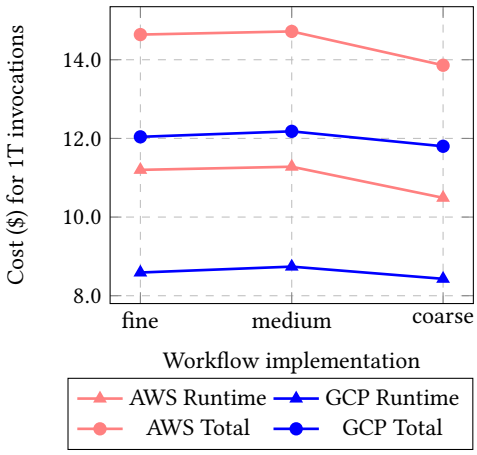



**Figure 3: Runtime cost and total cost for 1000 invocations on AWS and GCP.**

their order may primarily be attributed to the reduced number of file transfers in the coarse implementation. The discrepancy between the medium and the fine implementation may result from functions 9, 13, 15, and 18, each utilizing only a quarter of the memory compared to the other implementations.

*4.3.3 Analize the cost.* When evaluating runtime cost, the coarse implementation stands out as the most economical option followed by the fine and the medium implementations on both providers. The primary contributing factor is once again the number of file transfers, which allows the coarse implementation to utilize function deployments with equal or less memory compared to the other implementations, without decisively increasing the makespan. On AWS, the coarse implementation is 6.71 % cheaper than the fine, whereas, on GCP, the difference is less pronounced, with the coarse implementation only saving 1.90 % in costs compared to the fine implementation. We suspect that function 7-8 primarily contribute to this discrepancy, as AWS employs a 135 MB deployment compared to GCP's 1 GB and the function is executed on 141 instances in parallel. Examining the file transfer costs demonstrates that the coarse implementation saves 2.25 % compared to the fine and 2.02 % compared to the medium implementation. Since both AWS and GCP utilize identical pricing structures, this finding is applicable to both service providers. Evaluating the total cost reveals a 5.33 % saving on AWS when comparing the fine to the coarse and a 1.99 % cost saving on GCP.

Overall, the coarse implementation emerges as the most economical choice on both providers. On GCP, it even achieves Pareto optimality, dominating both the makespan and cost. Particularly for data-intensive workflows like Montage, bundling computations into single serverless functions, thereby reducing the necessity for file transfers, substantially enhances cost efficiency. However, the fact that the fine implementation is cheaper than the medium on both providers indicates the potential for separation with a higher granularity at the expense of a longer makespan.

### 4.4 Threats to validity

Whenever conducting experiments using public serverless providers it is important to highlight their opacity with respect to their hardware and software stack. This heterogeneity may lead to disparities in performance, as noted by Maissen *et al.* [30] and Copik *et al.* [11]. Moreover, Container-as-a-Service (CaaS) is reported to be a cheaper platform than FaaS for long running workflows and a hybrid approach with FaaS may circumvent service-specific limitations [8]. However, we used FaaS only since we evaluated a short-running version of montage, which mainly benefits from FaaS. Another contributing factor is the timeliness of the evaluation, as the cloud provider experiences load at different points in time, which affects performance, as reported by Kelly *et al.* [25]. Furthermore, our approach relies on the cost-performance ratio currently offered by the providers in the selected regions. It's worth mentioning that both AWS and GCP offer varied pricing for their services depending on the region. Therefore, repeating these experiments in different regions will likely yield divergent results. Finally, we ran five repetitions, which may be insufficient for long-running cloud applications, which can experience high variability in network traffic [46]. However, all evaluated Montage workflow implementations are running for a few minutes.

## 5 RELATED WORK

In this section we discuss various observations reported by the researchers specifically for the Montage workflow and how serverless functions, in general, are affected by various deployments of the functions with different memory.

### 5.1 Observations for the Montage workflow

Early research was conducted by Jackson *et al.* [20], who evaluated the applicability of scientific computing on AWS EC2. Humphrey *et al.* [19] investigated a hybrid architecture, using in-house computational resources alongside virtual machines on Microsoft Azure. Juve *et al.* [23] characterized various I/O reads and writes, peak memory, and CPU utilization of all Montage tasks. Balis [4] implemented Montage using the Hyperflow's data-flow approach and a high level of abstraction to run the workflow independently of the underlying runtime environment. However, all the aforementioned works used virtual machines as resources, in which the memory is not fully assigned to the workflow tasks, but rather it is managed by the guest operating system.

Hyperflow [31] evaluated that serverless implementations of Montage are cheaper than the classical serverfull implementations deployed on AWS EC2 virtual machines. However, the authors also reported high variability in performance for the functions that are nested in a parallel loop. Hautz *et al.* [17] reported that functions run the computing part and download data faster using AWS Lambda and S3, while GCP functions upload data faster on GCP cloud storage. The authors used the implementation in the Abstract Function Choreography Language [37] and executed the workflow with the xAFCL [38] serverless workflow management system. We used this implementation of the workflow in our evaluation as the medium implementation.

### 5.2 Observations for FaaS deployments

Jonas *et al.* [22] reported huge delays when invoking more functions simultaneously. Ristov *et al.* [38] reported that the overall round trip time is increased. Moreover, the spawn start affects the performance of the functions [36].

FaaSt [34] analyzed the performance of various cloud providers to the same data storage for BWA. Other researchers reported a speedup when collocating the functions closer to the data rather than moving data to the function [42, 43]. We followed the latter approach and always collocated the functions together with the storage within the same region.

SimLess [35] introduced a deterministic model for the overall round trip time of serverless functions that are deployed across multiple regions of the cloud provider. The model estimates the overall round trip time in one region from the executions in another region of the same function. SizeLess [15] used an ML approach to estimate the function execution time by learning from running the same function with 512 MB. However, both approaches consider a fixed function and workflow setup, while SimLess analyzed the same implementation of the Montage, but still with a different problem size.

## 6 CONCLUSION

In this paper, we conducted a series of experiments to investigate how different serverless implementations of the Montage workflow with various function deployments affect its makespan and cost. Our investigation led to two important observations.

We first observed a superlinear speedup for some functions when assigning them with more memory, which yielded that they dominated their compatriots with less memory, both for the performance and cost. This insight was mainly observed on AWS due to the fine-grained pricing model down to 1 ms compared to GCP's coarse-grained of 100 ms. With this simple method, we were able to reduce the overall makespan and cost by 67.27 % and 3.28 % on AWS, respectively. On GCP, a smaller improvement was observed with a 10.93 % reduction in makespan and a marginal reduction of 0.05 % in cost. Secondly, the coarse implementation improves both performance and cost by 24.98 % and 5.33 % on AWS, and 12.96 % and 1.99 % on GCP. Surprisingly, on AWS, the medium implementation achieved the smallest makespan, further reducing the time by 6.92 % compared to the coarse implementation. However, this improvement came at a disproportionately higher cost.

We believe that these results will be very valuable for the research community. We will further extend our work in several directions. We will first investigate other serverless workflows [6, 17] and with more fine-grained memory setups. Further on, we will develop a multi-objective scheduler that will determine the optimal setup for the workflows across different providers, especially since our evaluation reported that none of the evaluated providers dominates the other for all workflow functions. Finally, based on the decisions of the scheduler, we will automatize the process of fusion of workflow functions, their packaging, and deployment.

## ACKNOWLEDGEMENT

This research received funding from Land Tirol, under the contract F.35499.

# REFERENCES


[1] Ali Al-Haboobi and Gabor Kecskemeti. 2021. Improving Existing WMS for Reduced Makespan of Workflows with Lambda. In *Euro-Par 2020: Parallel Processing Workshops: Euro-Par 2020 International Workshops, Warsaw, Poland, August 24–25, 2020, Revised Selected Papers 26*. Springer, 261–272.

[2] Aitor Arjona, Pedro García López, Josep Sampé, Aleksander Slominski, and Lionel Villard. 2021. Triggerflow: Trigger-based orchestration of serverless workflows. *Future Generation Computer Systems* 124 (2021), 215–229. https://doi.org/10.1016/j.future.2021.06.004

[3] Y. Babuji, J. Bryan, R. Chard, K. Chard, I. Foster, B. Galewsky, D. S. Katz, and Z. Li. 2021. Federated Function as a Service for eScience. In *2021 IEEE 17th International Conference on eScience (eScience)*. IEEE Computer Society, Los Alamitos, CA, USA, 251–252. https://doi.org/10.1109/eScience51609.2021.00046

[4] Bartosz Balis. 2016. HyperFlow: A model of computation, programming approach and enactment engine for complex distributed workflows. *Fut. Gen. Comp. Syst.* 55 (2016), 147 – 162. https://doi.org/10.1016/j.future.2015.08.015

[5] G Bruce Berriman, Ewa Deelman, John C Good, Joseph C Jacob, Daniel S Katz, Carl Kesselman, Anastasia C Laity, Thomas A Prince, Gurmeet Singh, and Mei-Hu Su. 2004. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In *Optimizing scientific return for astronomy through information technologies*, Vol. 5493. SPIE, 221–232.

[6] Tekin Bicer, Xiaodong Yu, Daniel J. Ching, Ryan Chard, Mathew J. Cherukara, Bogdan Nicolae, Rajkumar Kettimuthu, and Ian T. Foster. 2022. High-Performance Ptychographic Reconstruction with Federated Facilities. In *Driving Scientific and Engineering Discoveries Through the Integration of Experiment, Big Data, and Modeling and Simulation*, Jeffrey Nichols, Arthur 'Barney' Maccabe, James Nutaro, Swaroop Pophale, Pravallika Devineni, Theresa Ahearn, and Becky Verastegui (Eds.). Springer International Publishing, Cham, 173–189.

[7] Sebastian Burckhardt, Badrish Chandramouli, Chris Gillum, David Justo, Konstantinos Kallas, Connor McMahon, Christopher S. Meiklejohn, and Xiangfeng Zhu. 2022. Netherite: Efficient Execution of Serverless Workflows. *Proc. VLDB Endow.* 15, 8 (apr 2022), 1591–1604. https://doi.org/10.14778/3529337.3529344

[8] Krzysztof Burkat, Maciej Pawlik, Bartosz Balis, Maciej Malawski, Karan Vahi, Mats Rynge, Rafael Ferreira da Silva, and Ewa Deelman. 2021. Serverless Containers – Rising Viable Approach to Scientific Workflows. In *2021 IEEE 17th International Conference on eScience (eScience)*. 40–49. https://doi.org/10.1109/eScience51609.2021.00014

[9] Benjamin Carver, Jingyuan Zhang, Ao Wang, Ali Anwar, Panruo Wu, and Yue Cheng. 2020. Wukong: A Scalable and Locality-Enhanced Framework for Serverless Parallel Computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*. ACM, 1–15. https://doi.org/10.1145/3419111.3421286

[10] Henri Casanova, Rafael Ferreira da Silva, Ryan Tanaka, Suraj Pandey, Gautam Jethwani, William Koch, Spencer Albrecht, James Oeth, and Frédéric Suter. 2020. Developing accurate and scalable simulators of production workflow management systems with wrench. *Future Generation Computer Systems* 112 (2020), 162–175.

[11] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. 2021. SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing. In *Proceedings of the 22nd International Middleware Conference* (Québec city, Canada) *(Middleware '21)*. Association for Computing Machinery, New York, NY, USA, 64–78. https://doi.org/10.1145/3464298.3476133

[12] Rafael Ferreira Da Silva, Rosa M. Badia, Venkat Bala, Debbie Bard, Peer-Timo Bremer, Ian Buckley, Silvina Caino-Lores, Kyle Chard, Carole Goble, Shantenu Jha, Daniel S. Katz, Daniel Laney, Manish Parashar, Frederic Suter, Nick Tyler, Thomas Uram, Ilkay Altintas, Stefan Andersson, William Arndt, Juan Aznar, Jonathan Bader, Bartosz Balis, Chris Blanton, Kelly Rosa Braghetto, Aharon Brodutch, Paul Brunk, Henri Casanova, Alba Cervera Lierta, Justin Chigu, Taina Coleman, Nick Collier, Iacopo Colonnelli, Frederik Coppens, Michael Crusoe, Will Cunningham, Bruno De Paula Kinoshita, Paolo Di Tommaso, Charles Doutriaux, Matthew Downton, Wael Elwasif, Bjoern Enders, Chris Erdmann, Thomas Fahringer, Ludmilla Figueiredo, Rosa Filgueira, Martin Foltin, Anne Fouilloux, Luiz Gadelha, Andy Gallo, Artur Garcia Saez, Daniel Garijo, Roman Gerlach, Ryan Grant, Samuel Grayson, Patricia Grubel, Johan Gustafsson, Valerie Hayot-Sasson, Oscar Hernandez, Marcus Hilbrich, AnnMary Justine, Ian Laflotte, Fabian Lehmann, Andre Luckow, Jakob Luettgau, Ketan Maheshwari, Motohiko Matsuda, Doriana Medic, Pete Mendygral, Marek Michalewicz, Jorji Nonaka, Maciej Pawlik, Loic Pottier, Line Pouchard, Mathias Putz, Santosh Kumar Radha, Lavanya Ramakrishnan, Sashko Ristov, Paul Romano, Daniel Rosendo, Martin Ruefenacht, Katarzyna Rycerz, Nishant Saurabh, Volodymyr Savchenko, Martin Schulz, Christine Simpson, Raul Sirvent, Tyler Skluzacek, Stian Soiland-Reyes, Renan Souza, Sreenivas Rangan Sukumar, Ziheng Sun, Alan Sussman, Douglas Thain, Mikhail Titov, Benjamin Tovar, Aalap Tripathy, Matteo Turilli, Bartosz Tuznik, Hubertus Van Dam, Aurelio Vivas, Logan Ward, Patrick Widener, Sean Wilkinson, Justyna Zawalska, and Mahnoor Zulfiqar. 2023. Workflows Community Summit 2022: A Roadmap Revolution. https://doi.org/10.5281/ZENODO.7750670

[13] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. 2008. The cost of doing science on the cloud: The Montage example. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. 1–12. https://doi.org/10.1109/SC.2008.5217932

[14] Juan J. Durillo, Radu Prodan, and Jorge G. Barbosa. 2015. Pareto tradeoff scheduling of workflows on federated commercial Clouds. *Simulation Modelling Practice and Theory* 58 (2015), 95–111. https://doi.org/10.1016/j.simpat.2015.07.001

[15] Simon Eismann, Long Bui, Johannes Grohmann, Cristina L Abad, Nikolas Herbst, and Samuel Kounev. 2021. Sizeless: Predicting the optimal size of serverless functions. *arXiv preprint arXiv:2010.15162* (2021).

[16] John L. Gustafson. 1988. Reevaluating Amdahl's Law. *Commun. ACM* 31, 5 (May 1988), 532–533. https://doi.org/10.1145/42411.42415

[17] Mika Hautz, Sashko Ristov, and Michael Felderer. 2023. Characterizing AFCL Serverless Scientific Workflows in Federated FaaS. In *International Workshop on Serverless Computing (WoSC '23)*. ACM, Bologna, Italy, 24–29. https://doi.org/10.1145/3631295.3631397

[18] Michael T Heath. 2018. *Scientific Computing: An Introductory Survey, Revised Second Edition*. SIAM.

[19] Marty Humphrey, Zach Hill, Catharine van Ingen, Keith Jackson, and Youngryel Ryu. 2011. Assessing the Value of Cloudbursting: A Case Study of Satellite Image Processing on Windows Azure. In *2011 IEEE Seventh International Conference on eScience*. 126–133. https://doi.org/10.1109/eScience.2011.26

[20] Keith R. Jackson, Krishna Muriki, Lavanya Ramakrishnan, Karl J. Runge, and Rollin C. Thomas. 2011. Performance and cost analysis of the Supernova factory on the Amazon AWS cloud. *Sci. Program.* 19, 2–3 (apr 2011), 107–119. https://doi.org/10.1155/2011/498542

[21] Aji John, Kristiina Ausmees, Kathleen Muenzen, Catherine Kuhn, and Amanda Tan. 2019. SWEEP: Accelerating Scientific Research Through Scalable Serverless Workflows. In *IEEE/ACM International Conference UCC Companion*. ACM, New Zealand, 43–50.

[22] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the cloud: Distributed computing for the 99%. In *Symposium on Cloud Computing*. 445–451.

[23] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. 2013. Characterizing and Profiling Scientific Workflows. *Fut. Gen. Comp. Syst.* 29, 3 (March 2013), 682–692. https://doi.org/10.1016/j.future.2012.08.015

[24] Daniel Kelly, Frank Glavin, and Enda Barrett. 2020. Serverless Computing: Behind the Scenes of Major Platforms. In *IEEE International Conference on Cloud Computing (CLOUD)*. 304–312. https://doi.org/10.1109/CLOUD49709.2020.00050

[25] Daniel Kelly, Frank Glavin, and Enda Barrett. 2020. Serverless Computing: Behind the Scenes of Major Platforms. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. 304–312. https://doi.org/10.1109/CLOUD49709.2020.00050

[26] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic Ephemeral Storage for Serverless Analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 427–444. https://www.usenix.org/conference/osdi18/presentation/klimovic

[27] Heng Li and Richard Durbin. 2010. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics* 26, 5 (2010), 589–595.

[28] Zhuozhao Li, Ryan Chard, Yadu Babuji, Ben Galewsky, Tyler J. Skluzacek, Kirill Nagaitsev, Anna Woodard, Ben Blaiszik, Josh Bryan, Daniel S. Katz, Ian Foster, and Kyle Chard. 2022. funcX: Federated Function as a Service for Science. *IEEE Trans. on Parallel and Distributed Systems* 33, 12 (2022), 4948–4963. https://doi.org/10.1109/TPDS.2022.3208767

[29] Philip Maechling, Ewa Deelman, Li Zhao, Robert Graves, Gaurang Mehta, Nitin Gupta, John Mehringer, Carl Kesselman, Scott Callaghan, David Okaya, Hunter Francoeur, Vipin Gupta, Yifeng Cui, Karan Vahi, Thomas Jordan, and Edward Field. 2007. *SCEC CyberShake Workflows—Automating Probabilistic Seismic Hazard Analysis Calculations*. Springer London, London, 143–163. https://doi.org/10.1007/978-1-84628-757-2_10

[30] Pascal Maissen, Pascal Felber, Peter Kropf, and Valerio Schiavoni. 2020. FaaSdom: A Benchmark Suite for Serverless Computing. In *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems* (Montreal, Quebec, Canada) *(DEBS '20)*. Association for Computing Machinery, New York, NY, USA, 73–84. https://doi.org/10.1145/3401025.3401738

[31] Maciej Malawski, Adam Gajek, Adam Zima, Bartosz Balis, and Kamil Figiela. 2020. Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions. *Future Generation Computer Systems* 110 (2020), 502–514. https://doi.org/10.1016/j.future.2017.10.029

[32] Roland Mathá, Sasko Ristov, Thomas Fahringer, and Radu Prodan. 2020. Simplified Workflow Simulation on Clouds based on Computation and Communication Noisiness. *IEEE Transactions on Parallel and Distributed Systems* 31, 7 (2020), 1559–1574. https://doi.org/10.1109/TPDS.2020.2967662

[33] Maciej Pawlik, Pawel Banach, and Maciej Malawski. 2020. Adaptation of workflow application scheduling algorithm to serverless infrastructure. In *Euro-Par 2019: Parallel Processing Workshops: Euro-Par 2019 International Workshops, Göttingen, Germany, August 26–30, 2019, Revised Selected Papers 25*. Springer, 345–356.

[34] Sashko Ristov and Philipp Gritsch. 2022. FaaSt: Optimize makespan of serverless workflows in federated commercial FaaS. In *2022 IEEE International Conference on*

*Cluster Computing (CLUSTER '22)*. IEEE, Heidelberg, Germany, 182–194. https://doi.org/10.1109/CLUSTER51413.2022.00032

[35] Sashko Ristov, Mika Hautz, Christian Hollaus, and Radu Prodan. 2022. SimLess: Simulate Serverless Workflows and Their Twins and Siblings in Federated FaaS. In *2022 ACM SoCC '22: ACM Symposium on Cloud Computing (SoCC '22)*. ACM, San Francisco, CA, USA, 323–339. https://doi.org/10.1145/3542929.3563478

[36] Sashko Ristov, Christian Hollaus, and Mika Hautz. 2022. Colder Than the Warm Start and Warmer Than the Cold Start! Experience the Spawn Start in FaaS Providers. In *Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed Systems (APPLIED '22)*. ACM, Salerno, Italy, 35–39. https://doi.org/10.1145/3524053.3542751

[37] Sasko Ristov, Stefan Pedratscher, and Thomas Fahringer. 2021. AFCL: An Abstract Function Choreography Language for serverless workflow specification. *Fut. Gen. Comp. Syst.* 114 (2021), 368 – 382.

[38] Sasko Ristov, Stefan Pedratscher, and Thomas Fahringer. 2023. xAFCL: Run Scalable Function Choreographies Across Multiple FaaS Systems. *IEEE Transactions on Services Computing* 16, 1 (2023), 711–723. https://doi.org/10.1109/TSC.2021.3128137

[39] Sasko Ristov, Radu Prodan, Marjan Gusev, and Karolj Skala. 2016. Superlinear speedup in HPC systems: Why and when?. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*. 889–898.

[40] Josep Sampe, Pedro Garcia-Lopez, Marc Sanchez-Artigas, Gil Vernik, Pol Roca-Llaberia, and Aitor Arjona. 2021. Toward Multicloud Access Transparency in Serverless Computing. *IEEE Soft.* 38, 1 (2021), 68–74. https://doi.org/10.1109/MS.2020.3029994

[41] J. Sampe, M. Sanchez-Artigas, G. Vernik, I. Yehekzel, and P. Garcia-Lopez. 2021. Outsourcing Data Processing Jobs with Lithops. *IEEE Transactions on Cloud Computing* (Nov. 2021), 1–1. https://doi.org/10.1109/TCC.2021.3129000

[42] Biswajeet Sethi, Sourav Kanti Addya, Jay Bhutada, and Soumya K. Ghosh. 2023. Shipping Code towards Data in an Inter-Region Serverless Environment to Leverage Latency. *J. Supercomput.* 79, 10 (mar 2023), 11585–11610. https://doi.org/10.1007/s11227-023-05104-7

[43] Christopher Peter Smith, Anshul Jindal, Mohak Chadha, Michael Gerndt, and Shajulin Benedict. 2022. FaDO: FaaS Functions and Data Orchestrator for Multiple Serverless Edge-Cloud Clusters. In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*. 17–25. https://doi.org/10.1109/ICFEC54809.2022.00010

[44] Ion Stoica and Scott Shenker. 2021. From Cloud Computing to Sky Computing. In *Workshop on Hot Topics in Operating Systems (HotOS '21)*. ACM, Ann Arbor, Michigan, 26–32. https://doi.org/10.1145/3458336.3465301

[45] Ali Tariq, Austin Pahl, Sharat Nimmagadda, Eric Rozner, and Siddharth Lanka. 2020. Sequoia: Enabling Quality-of-Service in Serverless Computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*. ACM, Virtual Event, USA, 311–327. https://doi.org/10.1145/3419111.3421306

[46] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. 2020. Is Big Data Performance Reproducible in Modern Cloud Networks?. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 513–527. https://www.usenix.org/conference/nsdi20/presentation/uta

[47] Ao Wang, Jingyuan Zhang, Xiaolong Ma, Ali Anwar, Lukas Rupprecht, Dimitrios Skourtis, Vasily Tarasov, Feng Yan, and Yue Cheng. 2020. INFINICACHE: exploiting ephemeral serverless functions to build a cost-effective memory cache. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*. USENIX Association, Santa Clara, CA, USA, 267–282.

[48] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the Curtains of Serverless Platforms. In *USENIX Annual Technical Conference*. Boston, MA, USA, 133–145.

[49] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, and Ion Stoica. 2023. SkyPilot: An Intercloud Broker for Sky Computing. In *Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX, Boston, MA, 437–455.

[50] Minchen Yu, Tingjia Cao, Wei Wang, and Ruichuan Chen. 2021. Restructuring Serverless Computing with Data-Centric Function Orchestration. *arXiv preprint arXiv:2109.13492* (2021).

# Towards a Workload Trace Archive for Metaverse Systems

Radu Apșan*
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
R.Apsan@student.vu.nl

Damla Ural*
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
D.Ural@student.vu.nl

Paul Daniëlse*
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
P.E.G.Danielse@student.vu.nl

Vlad-Andrei Cursaru*
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
V.Cursaru@student.vu.nl

Eames Trinh*
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
E.V.T.Trinh@student.vu.nl

Jesse Donkervliet*
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
J.J.R.Donkervliet@vu.nl

Alexandru Iosup
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
A.Iosup@vu.nl

## ABSTRACT

Recent years have seen a resurgence of societal interest in Metaverses and virtual reality (VR), with large companies such as Meta and Apple investing multi-billion dollars into its future. With the recent developments in VR hardware and software, understanding how to operate these systems efficiently and with good performance becomes increasingly important. However, studying Metaverse and VR systems is challenging because publicly available data detailing the performance of these systems is rare. Moreover, collecting this data is labor-intensive because VR devices are end-user devices that are driven by human input. In this work, we address this challenge and work towards a *workload trace archive* for Metaverse systems. To this end, we design, implement, and validate librnr, a system to record and replay human input on VR devices, automating large parts of the process of collecting VR traces. We use librnr to collect 106 traces with a combined runtime of 7 hours from state-of-the-art VR hardware under a variety of representative scenarios. Through analysis of our initial results, we find that power use of VR devices can increase by up to 29% depending on the location of the VR device relative to the user-defined play area, and show that noticeable performance degradation can occur when network bandwidth drops below 100 Mbps. Encouraging community adoption of both librnr and the emerging trace archive, we publish both according to FAIR data principles at https://github.com/atlarge-research/librnr.

## CCS CONCEPTS

• **Human-centered computing** → **Virtual reality**; • **Networks** → **Network measurement**.

*These authors contributed equally to this work.

This work is licensed under a Creative Commons Attribution-ShareAlike International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651421

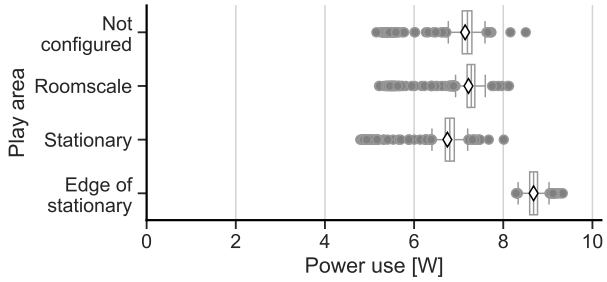



**Figure 1: Meta Quest Pro power use when using different play area configurations.**

## KEYWORDS

metaverse, virtual reality, online gaming, workload traces, performance analysis, real-world experiments, FAIR data, open science

**ACM Reference Format:**
Radu Apșan, Damla Ural, Paul Daniëlse, Vlad-Andrei Cursaru, Eames Trinh, Jesse Donkervliet, and Alexandru Iosup. 2024. Towards a Workload Trace Archive for Metaverse Systems. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651421

## 1 INTRODUCTION

Over the past decade, there has been an increasing and steady resurgence in interest in the topics of virtual reality (VR) and the *Metaverse*, with prominent examples including Meta's $36 billion investment towards building a metaverse [13], and Apple's entry into the VR market by releasing a $3,500 consumer VR device in February 2024 [4]. The concept of a Metaverse was introduced more than three decades ago [22], and, although community consensus on a definition is still lacking, presents a vision for a novel and fundamentally different way for people to interact with computers and each other. In this vision, many operations that today require a mouse and keyboard are performed using VR devices, which commonly consist of a head-mounted display (HMD) and a pair of hand-held controllers. *If successful, a metaverse holds great potential to benefit society at large across a wide range of application*

*domains.* For example, a metaverse can benefit medicine [6, 30], mental health [11], construction [2], and law enforcement [20] by improving professional collaborative environments.

For a metaverse to realize its potential and gain large-scale societal adoption, it must provide good performance and energy efficiency under a wide range of use cases. For example, the VR devices that users use to interact with a metaverse have stringent requirements on latency and energy efficiency. Specifically, devices must display a new frame to the user roughly every 14 ms to meet the performance requirement of 72 frames per second, and reduce the probability of motion sickness [25], while providing a battery life in the order of hours.

However, although there has been increasing interest in the topic, understanding the performance and overall behavior of VR systems remains challenging for two important reasons. First, there is a lack of *publicly available traces* that allow researchers to study the behavior of these systems. The collection and analysis of traces is a common and important approach in computer systems to improve understanding of system behavior, and has been successfully applied in a wide range of application domains including cloud computing [7], workflow scheduling [23], and online gaming [12]. Second, collecting traces for VR systems is labor-intensive because VR systems are end-user devices. Similar to smartphones, their realistic workloads consist of applications that respond to human inputs, which are challenging to automate realistically.

Although there exists work that focuses on the performance of metaverse-related technologies, these studies typically focus on the performance of individual subsystems, such as the motion-to-photon latency [27], or propose new techniques to improve performance, for example through computational offloading [8, 29].

In this work, we make an important step towards a *workload trace archive for metaverse systems*, allowing both researchers and developers to better understand the behavior of these state-of-the-art systems. For example, Figure 1 shows a simplified result based on traces collected as part of this work, and shows that the power use of VR device is affected by the play area and VR positioning. To this end, we design librnr, a novel open-source tool that simplifies and partially automates the collection of both user-input and performance-measurement traces. We use librnr to obtain traces from a variety of VR devices and deployment scenarios, and present surprising preliminary results on the behavior of state-of-the-art VR hardware.

Our key contributions are:

- **C1** We design and implement librnr, a novel system to record and replay user-input traces on state-of-the-art VR hardware (Section 3). librnr is designed to be compatible with most state-of-the-art VR applications out of the box, without modifying the application. librnr captures traces in an open format, allowing researchers to analyze user and system behavior.
- **C2** We validate, through real-world experiments, the accuracy and overhead of librnr, and show that librnr can replay human inputs with high accuracy without introducing significant performance overhead (Section 4).
- **C3** We use librnr to bootstrap a workload trace archive for metaverse systems (Section 5). We obtain VR system traces from two popular VR devices under a range of network conditions, while exposed to the same (human input) workload. We analyze our traces and present novel insights into the behavior of VR systems.
- **C4** We publish librnr, and the collected traces, as open-source artifacts according to FAIR research principles, on Github: https://github.com/atlarge-research/librnr/tree/trace_files_and_report/traces.

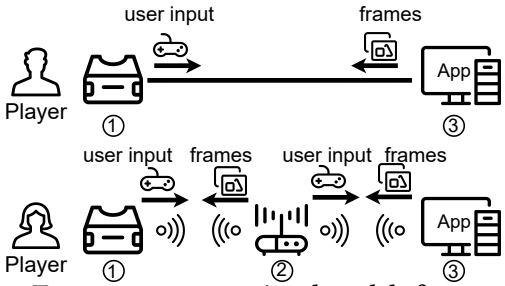



**Figure 2: Two common operational models for metaverse applications, linking a user's VR device (left) to a more powerful device (right) for computational offloading. In the two modes, control operations and rendered frames are sent via wire (top) or wirelessly (bottom), respectively.**

## 2 BACKGROUND

In this section, we present a client-centric model for modern metaverse systems and discuss important properties relevant to this work. We present a visual overview of our model in Figure 2.

Our model starts with a user (or player) of the metaverse system, who interacts with the system through a VR device (labeled ① in Figure 2). The VR device typically consists of a HMD and hand-held controllers. The HMD contains displays that show the user a three-dimensional virtual world. The HMD also tracks movements of itself and the hand-held controllers. Together with occasional user button-presses, these control commands (or *controls*) are sent from the VR device to the host.

The host (③) is a machine either close to the user or deployed in a cloud environment and is responsible for simulating and rendering the metaverse application (or *app*), and takes the received controls as inputs for its simulation. The app performs a simulation iteration and renders a new frame at a fixed rate. For VR applications, the frame rate is typically 72 or 90 Hz. The application must consistently produce frames at this rate to maintain good performance. Lower frame rates are noticeable to users and induce motion sickness [25]. Each rendered frame is sent back to the HMD to be displayed to the user, giving the user the illusion of being in a different space.

Depending on how the user sets up their system, controls and frames are sent via a wire or a WiFi router (②). Both deployment methods have advantages and are common. Because the app must respond to user controls in real-time, and frames must be delivered to the HMD at a high rate, data must be exchanged with both low latency (i.e., in the order of milliseconds) and high bandwidth (i.e., in the order of 100 Mbps). When using a wired setup, these requirements are relatively simple to guarantee. When using a WiFi router, meeting these requirements is more challenging, and depends on the quality of the WiFi router of the user. However, a wireless setup

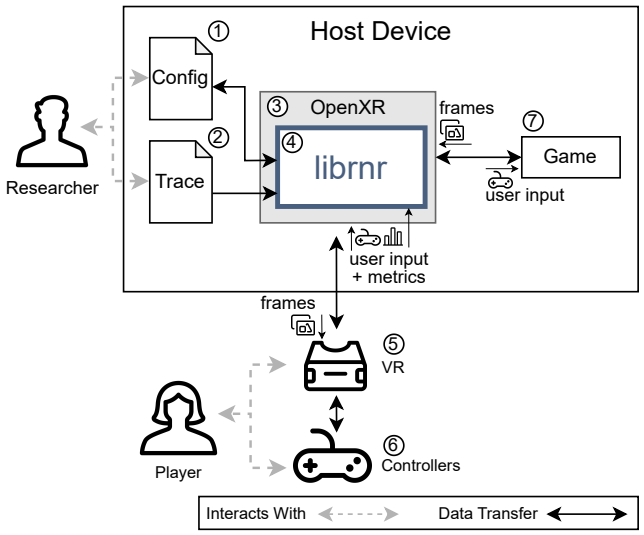



**Figure 3: Design overview of librnr.**

allows the user to move while using the device without worrying about the length of the cable.

## 3 DESIGN AND IMPLEMENTATION

In this section, we present the design of librnr. We discuss system requirements and give an overview of its design and implementation.

### 3.1 System Requirements

We identify here important system requirements for librnr:

- **R1** librnr's system traces must include important system-level metrics for VR applications, allowing users to analyze system behavior under different scenarios.
- **R2** librnr must support a wide range of state-of-the-art VR devices. To construct a workload trace archive that provides a large amount of diverse traces, librnr needs to support a wide variety of VR devices and software applications.
- **R3** While replaying user-input traces, the timing accuracy of events must be in the order of 10s of milliseconds, or up to approximately 10 frames, to ensure the resulting application and system behavior remain the same across replays.
- **R4** librnr should not introduce significant performance overhead. The traces recorded and replayed with librnr should be representative of real-world usage.

### 3.2 Design and Implementation Overview

Figure 3 presents our design for librnr. We design librnr (④) for two types of users: players and researchers. Players can use librnr to record VR input traces. Doing so only requires them to set librnr to *recording mode* in a configuration file (①). Afterwards, they can interact with the VR device as usual. Applications (⑦) typically obtain user-input information by polling the VR device for the location and orientation of the headset (⑤) and controllers (⑥), and button-press events. When the player starts an application, librnr starts intercepting these calls and writes the values to a trace file.

Researchers can use librnr to replay user-input traces (②) and collect system traces while controlling the environment (e.g., by limiting available network bandwidth) or the system under test (e.g., by using different VR headsets). This allows the researcher to obtain a large amount of system traces through user emulation, automating the most labor-intensive part of obtaining traces for VR systems. When playing back a trace, the researcher sets librnr to *replay mode* through its configuration file and starts the corresponding application. Once the application starts, it will start polling the VR device for user input. However, in replay mode, librnr will overwrite the values obtained from the VR device with values read from the user-input trace, sending previously recorded user inputs to the application.

In both recording and replaying mode, librnr captures important system-level metrics by sampling system performance counters on both the host and VR device at a frequency of 1 Hz (partially addresses Requirement R1). For both devices, these metrics include the information commonly available under the Linux `/proc` filesystem, while for the VR device librnr additionally collects metrics such as the number of frames per second, the amount of time spent on rendering frames, and battery usage data.

We implement librnr as an API layer in OpenXR. We choose OpenXR because it (③) is a modern, open-source standard and API, backed by the majority of major VR device manufacturers, which provides an abstraction layer between applications and the VR device (addresses Requirement R2). Such an abstraction is useful for two important reasons. First, it allows application developers to support a wide variety of VR devices without creating separate implementations of their applications. Second, it allows VR manufacturers to release new VR devices that are backwards-compatible with existing applications without additional engineering effort. OpenXR supports so-called API layers, which are side-loaded libraries that can intercept calls made through the OpenXR API.

## 4 VALIDATION OF LIBRNR

In this section, we present a preliminary validation of Requirements R3 and R4 from librnr's design (Section 3.1) through real-world experiments. We validate librnr using two setups, PC-A running the Meta Quest Pro (MQP) and PC-B running the Meta Quest 2 (MQ2). For the full experimental setup, see Section 5.1.

Overall, we find:

- **V1** librnr can replay traces with a median delay of 14 ms, or 1 frame, and the delay remains stable over time (Section 4.1). High timing accuracy addresses Requirement R3, and is important for creating reproducible input behavior.
- **V2** librnr does not introduce significant performance overhead (Section 4.2). Low performance-overhead addresses Requirement R4, and is important for collecting representative traces from metaverse systems.

### 4.1 Replay Timing Accuracy (Requirement R3)

We design librnr to enable researchers to better understand the behavior of different metaverse systems and environments. To this end, it is important that replaying the same input trace results in the same system workload. In this section, we validate this behavior through a real-world experiment in which we compare the timestamps of the events in the recorded trace with those obtained during replay. Overall, we find that librnr's timing is highly accurate, and meets Requirement R3. We visualize this result in Figure 4.

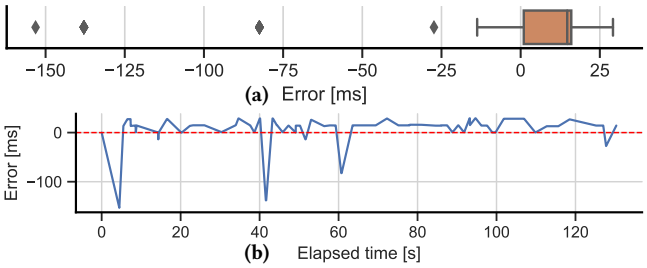



**Figure 4: Timing accuracy (i.e., error) of librnr as a statistical summary (top plot) and over time (bottom plot).**

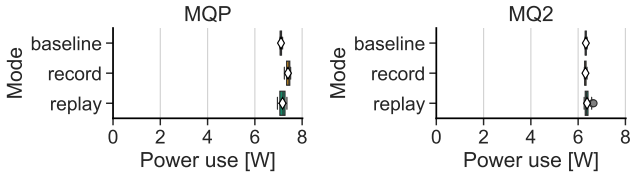



**Figure 5: librnr energy overhead on Meta Quest Pro (MQP) and Meta Quest 2 (MQ2).**

Figure 4a shows the inaccuracy of timed events while replaying a user-input trace, and reveals that the time difference (i.e., error) is mostly stable between 1 ms and 16 ms, or approximately 0 and 1 frames, based on a frame rate of 72 Hz. The horizontal axis shows the error between when an event was originally recorded, and when it is replayed by librnr. When the error is positive, it indicates that the replay is behind the original recording, and vice versa. Closer to zero is better. The box shows the 25$^{th}$, 50$^{th}$, and 75$^{th}$ percentiles, and the whiskers extend up to 1.5 times the inter-quartile range (IQR). The diamonds show all remaining outliers. The plot shows that half of the observed error values (inside the box) have an error between 1 ms and 16 ms, and that the median error value is 15 ms, or 1 frame.

However, the figure shows significant outliers of up to -153 ms, or -11 frames. To investigate these outliers, we visualize the system's behavior over time in Figure 4b. The horizontal axis shows the progression of time during the experiment, the vertical axis shows the error, and the blue curve shows the error over time. The figure shows that the spikes are both rare and isolated events and are caused by the slight error we see throughout the replays.

## 4.2 System Overhead (Requirement R4)

In this section, we present initial results towards validating the system overhead of librnr on both the VR and host devices using real-world experiments. Although we analyze overhead on power use, GPU utilization, and CPU utilization, we focus in this section on the former two. The results for the CPU utilization are similar to those observed for the GPU utilization.

Figure 5 shows the overhead of power consumption when using librnr with two highly-popular VR devices: the MQP and the MQ2. The figure shows that librnr's power-usage overhead is generally insignificant. The left and right plots in the figure show the power use for the MQP and MQ2 respectively. The horizontal axes show the power use of the VR device, and the vertical axes show the alternative experiment setups. *Baseline* indicates measurements performed without using librnr, whereas *record* and *replay* indicate measurements performed while librnr is recording or replaying a user-input trace, respectively. The boxes summarize the behavior

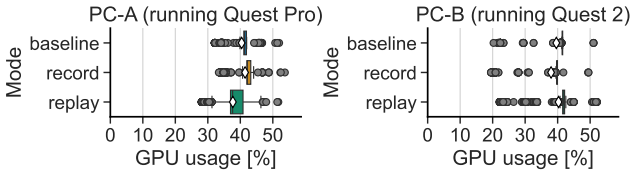



**Figure 6: librnr GPU-usage overhead on the two PCs. PC-A is the setup running Windows 11, PC-B is the setup running Windows 10. See §5.1 for exact specifications.**

**Table 1: Overview of experimental hardware specifications.**

| **Hardware** | **PC-A** | **PC-B** |
|---|---|---|
| OS | Windows **11** | Windows **10** |
| CPU | AMD Ryzen 5 7600X | AMD Ryzen 5 7600X |
| GPU | GeForce RTX **3080** | GeForce RTX **4070** |
| WiFi | 802.11ax | 802.11ax |
| **Headsets** | **MQ2** | **MQP** |
| Released | 2019 | 2022 |
| CPU | Snapdragon XR2 | Snapdragon XR2+ |
| Battery | 3640 mAh | 5348 mAh |
| WiFi | WiFi 6 | WiFi 6E |

across 6, 6, and 60 traces of approximately 4 minutes per trace for the baseline, record, and replay setups, respectively.

The figure shows that, in most configurations tested, the power use of the device is highly similar. Using librnr on the MQ2, the maximum difference between mean power usage across the baseline, record, and replay setups is 0.07 W. The setups also show similar distributions, with the maximum difference between the minimum and maximum power usage across setups on the MQ2 being 0.17 W.

Similar trends can be observed for the MQP. The maximum mean power usage difference across the three modes is 0.29 W, and the difference between the minimum and maximum power usage across setups on the MQP is 0.36 W.

Figure 6 shows librnr overhead on GPU utilization, and shows that librnr does not incur significant overhead in any of the setups tested. The figures show the GPU utilization when using librnr on the MQP (left) and the MQ2 (right). The horizontal axes show the GPU usage, and the vertical axes show the experiment setup. The boxes show the 25$^{th}$, 50$^{th}$, and 75$^{th}$ percentiles, and the box whiskers extend up to 1.5 times the IQR. Grey circles with a black border indicate outliers. From the figure, we observe that the distributions of GPU usage are highly similar across all setups. Specifically, the largest difference between the depicted percentiles is 3.84 percentage points, between 75$^{th}$ and 25$^{th}$ percentiles on the PC-A running MQP, with librnr in replay mode. We also observe a high number of outliers across all setups, ranging from 19.47% to 53.71%.

# 5 RESULTS

Using the user-input traces collected with librnr, we conduct preliminary real-world experiments on two highly popular VR devices across a variety of environments.

We summarize our experiments in Table 2, and derive the following **M**ain **F**indings:

**MF1** VR power use is significantly affected by the location of the VR and the user-defined play area (Section 5.2). Changing

**Table 2: Experiment overview.**

| Section | Experiment | Traces | | Bandwidth limit | Devices |
|---|---|---|---|---|---|
| | | Recordings | Replays | [Mbps] | |
| §5.2 | Effect of play area settings on energy use | 1 | 25 | - | PC-B + MQP, PC-B + MQ2 |
| §5.3 | Limiting bandwidth effects on VR | 2 | 24 | 30, 50, 80, 100 | PC-A + MQP, PC-B + MQ2 |
| §5.4 | VR performance comparison when offloading | 4 | 40 | - | PC-A + MQP, PC-A + MQ2 |

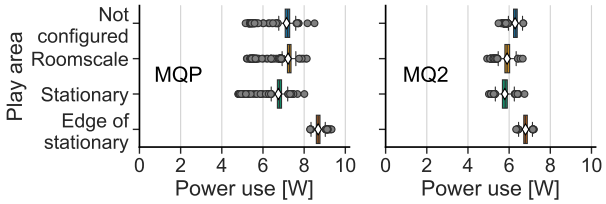



**Figure 7: Effect of play area settings on power consumption for the Quest Pro (MQP) and Quest 2 (MQ2) VR devices.**

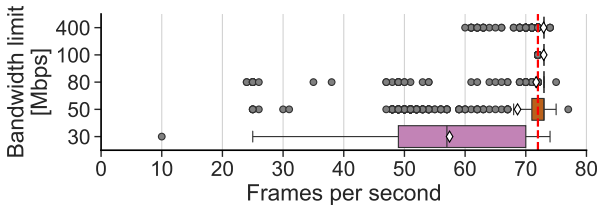



**Figure 8: Effect of bandwidth limits on Meta Quest 2 frames per second (FPS). Larger FPS is better, the red dotted line marks the minimum recommended frame rate (i.e. 72 FPS).**

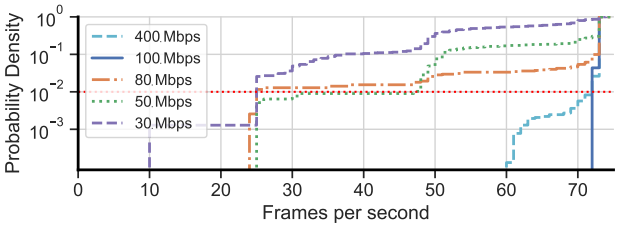



**Figure 9: Cumulative probability density function for the frames per second (FPS) on Meta Quest 2. Below the red dotted line is the 99th percentile (i.e., $10^{-2}$).**

the play area settings and VR location can increase mean power use by up to 29% and 17%, for the MQP and MQ2, respectively.

**MF2** Frame rate is significantly affected by available network bandwidth (Section 5.3). For example, user experience significantly deteriorates when using the MQ2 if network bandwidth is limited to 80 Mbps.

**MF3** There is no significant difference in performance between the MQP and MQ2 (Section 5.4). Although the MQP is a flagship device and the MQ2 a budget-friendly device, their performance is similar when offloading to a remote machine.

## 5.1 Experiment Setup

We conduct our experiments using two representative host devices and two popular VR devices, the Meta Quest Pro (MQP), a state-of-the-art VR device designed for professional applications [17], and Meta Quest 2 (MQ2), the best-selling VR device worldwide [5], whose hardware specifications are listed in Table 1. The VR devices are connected to a host device to offload application simulation and rendering. The video frames and user inputs are sent to and from the VR devices through a 5 GHz WiFi router, which provides a bandwidth of up to 1200 Mbps.

We use the application *Beat Saber* as our workload throughout our experiments. Games represent a significant part of the VR market [1], and Beat Saber is one of the best-selling VR games worldwide, with over 4 million copies sold.

## 5.2 VR Play Area Settings Affect VR Power Use

The play area settings and VR positioning significantly affect the power use of VR devices, increasing the mean power use by up to 29% when the VR is on the edge of the play area. The Oculus software allows two play area modes: stationary mode with a preset circular play area and roomscale mode which allows the user to draw the play area using the controller. Finally, there is the option to disable the play area setting [16] under developer options.

Figure 7 shows the energy consumption for two VR devices, the MQP and MQ2, under the different play area configurations. The horizontal axes show the power use of the device, and the vertical axes show the play area configurations used during the experiment. The boxes show the 25$^{th}$, 50$^{th}$, and 75$^{th}$ percentiles, and the white diamonds mark the mean. The lowest mean power use of the MQP is 6.7 W, when using a stationary play area with the VR device placed in the middle. Although changing the play area type to roomscale or turning it off does not significantly affect its power use, placing the VR on the edge of the stationary play area increases its power use by up to 29% to 8.6 W. We see a similar trend on the MQ2, consuming the least average power of 5.8 W for the stationary play area, with the average power increasing by up to 17% when moving the device to the stationary play area edge.

We find that this difference occurs due to the VR device blending the game frames with the live feed from the onboard cameras. This is a safety feature of the device that prevents the user from walking outside the user-determined play area. We recommend that for the most energy-efficient experience, users should remain in the center of the defined play area.

## 5.3 Limiting Bandwidth Leads to Sudden Performance Degradation

Increasingly limiting the network bandwidth available for streaming video from the host to the VR device leads to sudden performance degradation.

Figure 8 shows this result. The horizontal axis shows the number of frames per second, and the vertical axis shows the bandwidth limits used in the experiment. The boxes show the 25$^{th}$, 50$^{th}$, and 75$^{th}$ percentiles, and the white diamond marks the mean. The vertical red-dashed line indicates 72 frames per second, which is the target

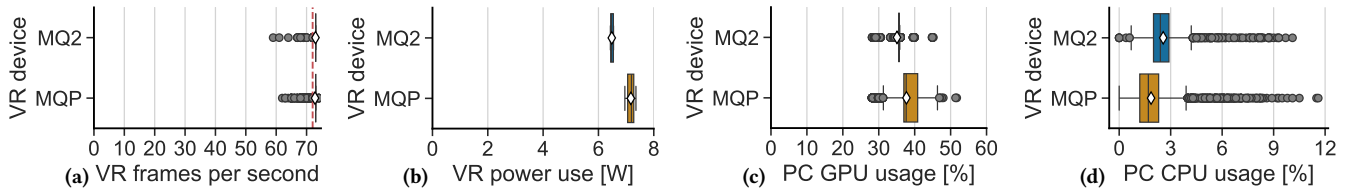



**Figure 10: Comparison in workload offloading between Meta Quest 2 (MQ2) and Meta Quest Pro (MQP) using the PC-A setup. See §5.1 for device specifications.**

frame rate. The plot shows that for bandwidth limits of 400 Mbps and 100 Mbps, overall performance meets the 72 frames per second (FPS) target, with outliers down to 60 FPS. However, bandwidth limits of 80 Mbps and below introduce significantly lower outliers, down to 10 FPS (!) for a bandwidth limit of 30 Mbps.

Figure 9 presents a reverse cumulative distribution function (CDF) of the same data, allowing us to analyze the tail of the distribution in more detail. The horizontal axis shows the frames per second (FPS), and the horizontal axis shows the fraction of FPS samples. The horizontal red-dashed line indicates the 99$^{th}$ percentile. The 99$^{th}$ percentile is relevant because the system must maintain stable performance over time. The figure shows that for both 400 Mbps and 100 Mbps bandwidth limits, the 99$^{th}$ percentile is 70 FPS and 72 FPS respectively (the corresponding curves intersecting the red dashed line). However, for bandwidth limits of 80 Mbps and below, the performance is significantly worse, with the 99$^{th}$ percentile frame rate being 25 FPS, 25.9 FPS, and 25 FPS for bandwidth limits of 30 Mbps, 50 Mbps, and 80 Mbps respectively. This sudden drop in performance indicates that existing video streaming software for VR systems imposes a clear lower limit on available network bandwidth.

## 5.4 VR Performance Similar across Devices When Offloading

In this experiment, we compare the performance of the MQP and MQ2. We find that, when offloading the application to a remote machine, the performance of both devices is highly similar. This result is shown in Figure 10.

Figure 10a shows that the performance of both devices is highly similar. The horizontal axis shows the number of frames per second (FPS) displayed to the user, and the vertical axis shows the two VR devices. The vertical red-dashed line shows the performance target of 72 FPS. The plots show that for both the MQ2 and the MQP, the performance is stable at 72 FPS, with outliers down to roughly 60 FPS. Because the frame rates are highly similar, and both devices have roughly the same screen resolution, this indicates that the visual quality on both devices is similar.

Figure 10b shows that the power use of both devices is comparable under this setup. The horizontal axis shows the power use of the VR device, and the vertical axis shows the two VR devices. For both devices, the power usage is stable, with the maximum inter-quartile range (IQR) being 0.2 W, and no outliers outside 1.5 times the IQR. The mean power use is 7.2 W and 6.5 W for the MQP and the MQ2 respectively. Although the MQ2 performs 10% better, we conclude that the energy efficiency of both devices is similar.

Figures 10c and 10d show the GPU and CPU usage of the offloading target (i.e., PC-A). The figures show that the offloading target requires a similar amount of resources to simulate and render the offloaded application. On the horizontal axes, the figures show the GPU and CPU usage respectively. On the vertical axes, the figures show the two VR devices. For both resources, the usage is similar across devices. The GPU usage is 39% and 35% for the MQP and MQ2 respectively, and the CPU usage is 2% and 2.6% for the MQP and MQ2 respectively. The reported CPU usage is the mean across all 12 virtual cores. Based on this result, we conclude that both VR devices have similar requirements for offloading target devices.

# 6 RELATED WORK

Much related work in the VR research community analyzes device performance under different conditions. Relatively to the body of prior work, we focus on the complex interplay between VR offloading, network conditions, performance, and energy use. Additionally, we publish the collected traces and a novel system to partially automate VR trace collection.

*Offloading.* Q-VR is a novel system that reduces frame latency by partially offloading frame rendering [26]. Similarly, Danhier *et al.* design and implement a benchmark for video rendering for VR devices using a custom rendering pipeline [9]. Several other studies exist that investigate the impact of computational offloading across the cloud continuum (e.g., differences between edge, local, and cloud) [21, 29]. However, in contrast with this work, these studies do not investigate the effects of network conditions on metrics such as energy use when offloading to a local device.

*Network Conditions.* There are multiple studies that focus on analyzing the network traffic and usage patterns associated with various VR applications, such as video games [10, 15, 28] and social VR platforms [3]. These studies typically focus on performance [3, 10], user experience [15], and modeling network phenomena [10, 28]. We extend existing work by collecting a broad range of important metrics to characterize the effect that network conditions have on performance, energy use, and overall system behavior.

*Energy Efficiency.* In the field of energy-efficiency research, studies have investigated the energy use of gaming PCs [18, 19], VR devices [14, 18], cloud gaming [18], and the metaverse [24]. Our work is novel in its evaluation of the interplay between performance, energy use, the system under test, and the deployment environment (e.g., network conditions), and the publication of traces collected in these different scenarios.

# 7 CONCLUSION AND ONGOING WORK

A metaverse, if successfully implemented, promises to fundamentally change the way people interact with computers. To this end, the industry is investing tens of billions of dollars to develop new VR hardware and software, which are predicted to be the main way

for users to interact with the metaverse. However, understanding the performance of VR devices and their surrounding ecosystem in practice remains challenging due to the lack of publicly available system traces. In this work, we address this challenge by making an important step towards a publicly-accessible workload trace archive for metaverse systems. To this end, we design and implement librnr, a novel tool that partially automates the collection of system traces. Through real-world experiments, we validate librnr and study the behavior of two VR devices: the Meta Quest 2, the best-selling VR device worldwide, and the Meta Quest Pro, a state-of-the-art VR device designed for professional applications. Through our experiments, we find that play area placement can increase VR power usage by up to 1.9 W, and that streaming video to our VR devices with good performance requires at least 80 Mbps of bandwidth. Throughout our experiments, we collect a total of 11 input traces with a total duration of 40 minutes, and 112 system traces with a total duration of 7 hours.

This article is part of ongoing work on a workload trace archive for metaverse systems. To this end, we are working to add support for reproducibility packages to librnr, and are collecting traces for additional (types of) applications, VR devices, and deployments.

## ACKNOWLEDGMENTS

This work is funded by a National Growth Fund through the Dutch 6G flagship project "Future Network Services," the projects NWO OffSense (OCENW.KLEIN.209), EU Graph-Massivizer, and CloudStars, and by structural funds from VU Amsterdam. We thank Joshua Offermans for his exploration of synthetically generated user-input traces.

## REFERENCES

[1] 2024. Virtual Reality in Gaming Market Size | Global Analysis [2028]. https://www.fortunebusinessinsights.com/industry-reports/virtual-reality-gaming-market-100271 Accessed 2024-02-06.

[2] Pooya Adami, Patrick B. Rodrigues, Peter J. Woods, Burcin Becerik-Gerber, Lucio Soibelman, Yasemin Copur-Gencturk, and Gale M. Lucas. 2021. Effectiveness of VR-based training on improving construction workers' knowledge, skills, and safety behavior in robotic teleoperation. *Adv. Eng. Informatics* 50 (2021), 101431.

[3] Ahmad Alhilal, Kirill A. Shatilov, Gareth Tyson, Tristan Braud, and Pan Hui. 2023. Network Traffic in the Metaverse: The Case of Social VR. In *43rd IEEE International Conference on Distributed Computing Systems, ICDCS 2023 - Workshops, Hong Kong, July 18-21, 2023*. 109–114.

[4] Apple. 2023. Introducing Apple Vision Pro: Apple's first spatial computer. https://www.apple.com/newsroom/2023/06/introducing-apple-vision-pro-apples-first-spatial-computer Accessed 2024-01-04.

[5] Martin Armstrong. 2023. Chart: Meta Leads the Way in VR Headsets. https://www.statista.com/chart/29398/vr-headset-kpis/ Accessed 2024-03-06.

[6] Patrick Carnahan, John Moore, Daniel Bainbridge, Gavin Wheeler, Shujie Deng, Kuberan Pushparajah, Elvis C. S. Chen, John M. Simpson, and Terry M. Peters. 2020. Applications of VR medical image visualization to chordal length measurements for cardiac procedures. In *Medical Imaging 2020: Image-Guided Procedures, Robotic Interventions, and Modeling, Houston, TX, USA, February 15-20, 2020*, Vol. 11315. 1131528.

[7] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. 153–167.

[8] Vittorio Cozzolino, Leonardo Tonetto, Nitinder Mohan, Aaron Yi Ding, and Jörg Ott. 2023. Nimbus: Towards Latency-Energy Efficient Task Offloading for AR Services. *IEEE Trans. Cloud Comput.* 11, 2 (2023), 1530–1545.

[9] Martin Danhier, Karim El Khoury, and Benoît Macq. 2023. An Open-Source Fine-Grained Benchmarking Platform for Wireless Virtual Reality. In *Virtual Reality and Mixed Reality - 20th EuroXR International Conference, EuroXR 2023, Rotterdam, The Netherlands, November 29 - December 1, 2023, Proceedings*, Vol. 14410. 115–121.

[10] Jesse Donkervliet, Matthijs Jansen, Animesh Trivedi, and Alexandru Iosup. 2023. Can My WiFi Handle the Metaverse? A Performance Evaluation Of Meta's Flagship Virtual Reality Hardware. In *Proceedings of the International Conference on Performance Engineering, Coimbra, Portugal, April, 2023*.

[11] Paul M.G. Emmelkamp and Katharina Meyerbröker. 2021. Virtual Reality Therapy in Mental Health. *Annual Review of Clinical Psychology* 17, 1 (2021), 495–519.

[12] Yong Guo and Alexandru Iosup. 2012. The Game Trace Archive. In *11th Annual Workshop on Network and Systems Support for Games, NetGames 2012, Venice, Italy, November 22-23, 2012*. 1–6.

[13] Business Insider. 2022. Charts: Meta's Metaverse Spending Losses, Reality Labs, VR, Mark Zuckerberg. https://www.businessinsider.com/charts-meta-metaverse-spending-losses-reality-labs-vr-mark-zuckerberg-2022-10 Accessed 2024-01-04.

[14] Yue Leng, Jian Huang, Chi-Chun Chen, Qiuyue Sun, and Yuhao Zhu. 2020. Energy-Efficient Video Processing for Virtual Reality. *IEEE Micro* 40, 3 (2020), 30–36.

[15] Yen-Chun Li, Chia-Hsin Hsu, Yu-Chun Lin, and Cheng-Hsin Hsu. 2020. Performance Measurements on a Cloud VR Gaming Platform. In *QoEVMA'20: Proceedings of the 1st Workshop on Quality of Experience (QoE) in Visual Multimedia Applications, Seattle, WA, USA, October 16, 2020*. 37–45.

[16] Meta. 2023. Set up your boundary for Meta Quest. https://www.meta.com/en-gb/help/quest/articles/in-vr-experiences/oculus-features/boundary/ Accessed 2024-03-13.

[17] Meta. 2024. Meta Quest Pro: Premium mixed reality. https://www.meta.com/quest/quest-pro/ Accessed 2024-03-13.

[18] Evan Mills, Norman Bourassa, Leo Rainer, Jimmy Mai, Arman Shehabi, and Nathaniel Mills. 2019. Toward Greener Gaming: Estimating National Energy Use and Energy Efficiency Potential. *Comput. Games J.* 8, 3-4 (2019), 157–178.

[19] Nathaniel Mills and Evan Mills. 2016. Taming the energy use of gaming computers. *Energy Efficiency* 9, 2 (2016), 321–338.

[20] Markus Murtinger, Jakob Carl Uhl, Helmut Schrom-Feiertag, Quynh Nguyen, Birgit Harthum, and Manfred Tscheligi. 2022. Assist the VR Trainer - Real-Time Dashboard and After-Action Review for Police VR Training. In *IEEE International Conference on Metrology for Extended Reality, Artificial Intelligence and Neural Engineering, MetroXRAINE 2022, Rome, Italy, October 26-28, 2022*. 69–74.

[21] Baraka William Nyamtiga, Airlangga Adi Hermawan, Yakub Fahim Luckyarno, Tae-Wook Kim, Deok-Young Jung, Jin Sam Kwak, and Ji-Hoon Yun. 2022. Edge-Computing-Assisted Virtual Reality Computation Offloading: An Empirical Study. *IEEE Access* 10 (2022), 95892–95907.

[22] Neal Stephenson. 1992. *Snow Crash*. Bantam Spectra Books.

[23] Laurens Versluis, Roland Mathá, Sacheendra Talluri, Tim Hegeman, Radu Prodan, Ewa Deelman, and Alexandru Iosup. 2020. The Workflow Trace Archive: Open-Access Data From Public and Private Computing Infrastructures. *IEEE Trans. Parallel Distributed Syst.* 31, 9 (2020), 2170–2184.

[24] Ștefan Vlăduțescu and Georgiana Camelia Stănescu. 2023. Environmental Sustainability of Metaverse: Perspectives from Romanian Developers. *Sustainability* 15, 15 (2023).

[25] Jialin Wang, Rongkai Shi, Wenxuan Zheng, Weijie Xie, Dominic Kao, and Hai-Ning Liang. 2023. Effect of Frame Rate on User Experience, Performance, and Simulator Sickness in Virtual Reality. *IEEE Trans. Vis. Comput. Graph.* 29, 5 (2023), 2478–2488.

[26] Chenhao Xie, Xie Li, Yang Hu, Huwan Peng, Michael B. Taylor, and Shuaiwen Leon Song. 2021. Q-VR: system-level design for future mobile collaborative virtual reality. In *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*. 587–599.

[27] Jingbo Zhao, Robert S. Allison, Margarita Vinnikov, and Sion Jennings. 2017. Estimating the motion-to-photon latency in head mounted displays. In *2017 IEEE Virtual Reality, VR 2017, Los Angeles, CA, USA, March 18-22, 2017*. 313–314.

[28] Sihao Zhao, Hatem Abou-zeid, Ramy Atawia, Yoga Suhas Kuruba Manjunath, Akram Bin Sediq, and Xiao-Ping Zhang. 2021. Virtual Reality Gaming on the Cloud: A Reality Check. In *IEEE Global Communications Conference, GLOBECOM 2021, Madrid, Spain, December 7-11, 2021*. 1–6.

[29] Ziehen Zhu, Xianglong Feng, Zhongze Tang, Nan Jiang, Tian Guo, Lisong Xu, and Sheng Wei. 2022. Power-efficient live virtual reality streaming using edge offloading. In *Proceedings of the 32nd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV 2022, Athlone, Ireland, 17 June 2022*. 57–63.

[30] Paul Zikas, Antonis Protopsaltis, Nick Lydatakis, Mike Kentros, Stratos Geronikolakis, Steve Kateros, Manos Kamarianakis, Giannis Evangelou, Achilleas Filippidis, Eleni Grigoriou, Dimitris Angelis, Michail Tamiolakis, Michael Dodis, George Kokiadis, John Petropoulos, Maria Pateraki, and George Papagiannakis. 2023. MAGES 4.0: Accelerating the World's Transition to VR Training and Democratizing the Authoring of the Medical Metaverse. *IEEE Computer Graphics and Applications* 43, 2 (2023), 43–56.

# Towards Geo-Distributed Training of ML Models in a Multi-Cloud Environment

Chetan Phalak
TCS Research, India
chetan1.phalak@tcs.com

Dheeraj Chahal
TCS Research, India
d.chahal@tcs.com

Manju Ramesh
TCS Research, India
manju.ramesh1@tcs.com

Rekha Singhal
TCS Research, India
rekha.singhal@tcs.com

## ABSTRACT

Geo-distributed (GD) training is a machine-learning technique that uses geographically distributed data for model training. Like Federated Learning, geo-distributed machine learning can provide data privacy and also benefit from the cloud infrastructure provided by many vendors in multiple geographies. However, GD training suffers from multiple challenges such as performance degradation due to cross-geography low network bandwidth and high cost of deployment. Additionally, all major cloud vendors such as Amazon AWS, Microsoft Azure, and Google Cloud Platform provide services in several geographies. Hence, finding a high-performance as well as cost-effective cloud service provider and service for GD training is a challenge. In this paper, we present our evaluation of the performance and cost associated with training models in multi-cloud and multi-geography. We evaluate multiple deployment architectures using computing and storage services from multiple cloud vendors. The use of serverless instances in conjunction with virtual machines for model training is evaluated in this study. Additionally, we build and evaluate cost models for estimating the cost of distributed training of models in a multi-cloud environment. Our study shows that the judicious selection of cloud services and architecture might result in cost and performance gains.

## CCS CONCEPTS

• **Computing methodologies** → **Model verification and validation**; • **Computer systems organization** → **Cloud computing**.

## KEYWORDS

Geo-distributed training, multi-cloud, cost model

**ACM Reference Format:**
Chetan Phalak, Dheeraj Chahal, Manju Ramesh, and Rekha Singhal. 2024. Towards Geo-Distributed Training of ML Models in a Multi-Cloud Environment. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651422

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651422

## 1 INTRODUCTION

Many large enterprises have their data servers located across the globe to store their customer data. The conventional way of training models in such scenarios involves collecting data at one data center by transmitting over a wide area network (WAN). However, the prevailing government regulations such as the General Data Protection Regulation (GDPR) enforce user data protection by restricting enterprises from owning data rights [33]. Federated learning [15] (FL) has emerged as a popular solution to address the problem of data islands while preserving data privacy.

GD training based on FL involves training models using data residing in multiple geographies. The need for GD training originates from the distribution and partitioning of data in different regions of the globe to preserve data privacy, government regulations, compliance laws, etc. One of the popular approaches for distributed learning is FedAvg [22] which involves local model training by each client. All participating clients share the gradients with the server which are aggregated and communicated back to each client. Local clients use aggregated gradients to update their models. The model training is a compute-intensive process and requires dedicated resources such as Virtual Machines (VMs) or bare-metal machines via IaaS offering on cloud. However, GD training imposes multiple challenges such as

- Cost escalations due to frequent model sharing across geographies. Additionally, performance degradation and fluctuations are expected as communication overwhelms the low bandwidth of WAN between participating locations [12].
- Capacity planning is a challenge due to heterogeneity in data sizes distributed across multiple locations. An optimal distributed training architecture and placement of resources such as client VMs, and storage services in various geographies is necessary for performance and cost advantages.

These challenges can be mitigated by judiciously choosing cloud resources and services available from cloud vendors. All popular cloud vendors have unique features such as configurations and cost models for the corresponding services provided by them [26] resulting in diverse performance and cost for a given workload.

All cloud service providers share cost models for their services. However, multi-cloud deployment of an application results in complex cost models. A robust cost for a multi-cloud deployment can result in estimating the expenses for multiple deployment scenarios.

As discussed above, aggregators perform computations sporadically when gradients are available from all the clients. Serverless instances are good options for running aggregators due to *pay-as-you-go* cost model. Major Cloud Service Providers (CSPs) provide serverless platforms known as Function-as-a-Service (FaaS) such as AWS Lambda [3], Microsoft Azure functions [23], and Google functions [10]. FaaS billing is based on *pay-as-you-go* such that you pay

only for the actual uses of the resources, unlike IaaS where users are charged for running instances even if they are idle. However, there are a few limitations of serverless instances such as (a) *peer-to-peer* (P2P) communication is not possible between serverless instances and (b) Serverless instances do not have persistent storage. These challenges can be addressed in FaaS using cloud storage services such as AWS S3 [4], and Google Cloud Platform (GCP) storage [9].

Further improvements in performance and cost are possible by using a hierarchical design of aggregators [5, 19, 29]. An intermediate layer is added to aggregate the gradients of the workers from a region or geography before performing a global aggregation.

An in-depth understanding of the performance and cost of GD training architecture in a multi-cloud environment is essential to build an efficient system. Also, apriori knowledge of the system performance and cost of deployment would be advantageous for making judicious decisions. In this work, we analyze GD training architectures using IaaS, FaaS, and storage services from AWS, Microsoft Azure, and Google Cloud Platform (GCP). We also propose a and cost model for estimating the performance and cost of GD training in a multi-cloud and multi-geography environment. Succinctly, our contribution is as follows

- We evaluate multiple Geo-distributed training architectures using IaaS and FaaS services from multiple-cloud vendors.
- We study a hierarchical architecture for aggregating the gradients using a serverless platform. An investigation of the impact on performance and cost due to the placement of the aggregator in a particular geography is presented.
- We present a model for estimating the cost of training using multiple cloud services in a multi-cloud environment.

The rest of the paper is structured as follows. We discuss related work in Section 2. Our architecture and its evaluation are discussed in Section 3. We discuss our cost model in Section 4. The experimental setup and analysis are presented in Sections 5 and 6 respectively followed by the conclusion in Section 7.

## 2 RELATED WORK

Geo-distributed training is being explored by researchers in both academia and industry [1] [12]. Several distributed training frameworks employing data parallelism, like Horovod [30] and HOGWILD [27], have been created.. An efficient communication library for distributed training of deep learning (DL) models in a public cloud cluster is presented in [31]. Most of the large-scale distributed training frameworks are designed for data distributed within a data center or a region. In very recent work, a framework called Multi-FedLS is proposed for reducing execution time and managing resources on a cloud for Federated Learning applications [6]. The framework provides insights into VM instance selection in multi-cloud but does not consider FaaS serverless platforms.

Previous research has investigated gradient aggregation techniques employing parameter servers [16] and all-reduce methods [18]. A GD training framework called Cloudless-Training based on a parameter server-based approach is presented in [32]. Another framework known as sky computing is designed to accelerate GD computing in a federated learning [35]. Distributed GraphLab [20] is an extension of GraphLab [21] and directly targets asynchronous, dynamic, graph-parallel computation in the shared-memory setting, which leads to network congestion reduction in distributed ML training. However, the mentioned frameworks focus on performance improvement in terms of latency and resource utilization in multiple geographies but do not consider multi-cloud environments. The acceleration of communication in a LAN and WAN environment for geo-distributed learning is presented in [1]. However, the work is focused on geo-distributed training using a single cloud data center in different geographies.

A framework called COSTA [7] is designed for cost monitoring and managing the workload migration to the public cloud. The placement of parameter servers in a wide-area network for geo-distributed machine learning is discussed in [17]. Finding the optimal data storage service in a multi-cloud environment using optimization algorithms is presented in [28]. To the best of our knowledge, there is no prior study on performance comparison cost estimation for ML training using IaaS as well as FaaS cloud services in a multi-cloud and multiple geography environment.

Nebula-I [34] is a framework for collaboratively training DL models over remote heterogeneous clusters specifically GPU and NPU connected via low-bandwidth. Nebula-I successfully helps to launch training tasks over cloud but training the general model is still a challenge. A geo-distributed ML system called Gaia [13] decouples the intra and inter-communication between data centers located in various geographies enabling different communication and consistency models for each. The advantages of multi-cloud deployment for AI workflows have been studied in [24–26]. However, it was primarily for inference workload which had different characteristics as compared to the long-running model training process.

Although few frameworks have been proposed for GD training, very little is known about the performance and cost implications in a serverless and multi-cloud environment. We believe that this work is a pioneering effort to study the performance and cost trade-offs in training models using IaaS, FaaS, and storage services from multiple cloud vendors.

## 3 OUR ARCHITECTURE

In this section, we discuss the proposed architecture. As shown in Fig. 1, GD training involves following steps

- (Step 1) Workers spread over multiple geographies or regions fetch training data from storage and train the model on a mini-batch from the data.
- (Step 2) On completion of the mini-batch, by all participating workers share gradients or local models with the aggregator or parameter server.
- (Step 3) The aggregator collects the data and sends back the aggregated gradients or model to all the workers.
- (Step 4) Workers update their models and continue with the next mini-batch of the data.

Training in Gradient Descent (GD) can be classified into *centralized* and *de-centralized* architectures. In a centralized architecture, all the workers from different geographies send their model or gradients to one master located in one of the geographies (Fig. 2).

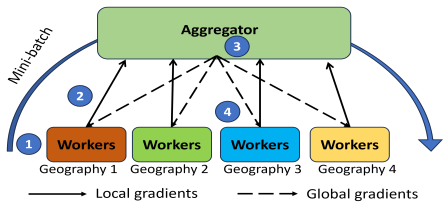



**Figure 1: GD training architecture**

However, in a decentralized architecture, local updates are aggregated in each geography and then also shared with the master for global updates (Fig.2).

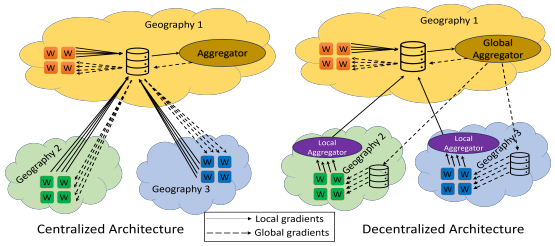



**Figure 2: (a) Centralized architecture for GD training (b) Decentralized architecture for GD training**

We use LambdaML [14] framework developed for training small ML/DL models in a distributed environment using a cloud serverless platform and storage services specifically in AWS. We have modified the original LambdaML framework to run workers on VM for training large models as well as aggregators using serverless instances in conjunction with storage services. Our modified implementation of LamdaML supports a multi-cloud environment allowing worker VMs, aggregator, and storage services to run on AWS, Azure, and Google cloud platforms. We continue using serverless instances for aggregation for cost savings and high scalability.

In our evaluation, we use an iterative training procedure called FedAvg [22]. Each participating worker trains the model locally and communicates its gradients to the serverless aggregator via cloud storage service. The server aggregates the gradients received from all the clients and synchronizes with all the workers in various geographies.

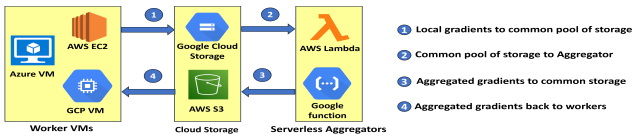



**Figure 3: Our architecture for multi-cloud GD training**

Our multi-cloud GD training deployment architecture, illustrated in Fig. 3, involves initializing VM instances across various geographies and clouds (AWS, Azure, and GCP) based on data storage service locations and their respective CSPs. Each VM instance acts as a worker, retrieving training data from associated cloud block storage services like AWS S3, Azure storage, or GCP storage. Workers share gradients with an aggregator via cloud storage service (GCP storage or AWS S3) after completing one mini-batch of training data. A serverless instance (GCP function or AWS Lambda) handles gradient aggregation due to restrictions on persistent storage and P2P communication in FaaS. The aggregated gradients are then saved back into the storage service, and all workers update their local models accordingly. This process repeats until convergence.

## 4 COST MODEL

In order to estimate the total cost of our architecture in a multi-cloud environment for GD training, we developed cost model. The total cost $C$ of multi-cloud geo-distributed training includes the following components - cost for compute ($C_{Compute}$) incurred by workers and aggregator, cost for storing the gradients in storage service ($C_{Storage}$) and the cost for data transfer ($C_{DataTransfer}$). All the rates are cloud vendor-specific and vary with each region.

$$C = C_{Compute} + C_{Storage} + C_{DataTransfer} \tag{1}$$

### 4.1 Compute Cost

Compute cost ($C_{Compute}$) includes cost due to workers VMs ($C_{IaaS}$) and aggregator or FaaS ($C_{FaaS}$)

$$C_{Compute} = C_{IaaS} + C_{FaaS} \tag{2}$$

*Compute cost for worker VMs ($C_{IaaS}$):* The worker VMs are billed for the entire duration for which it is used. The cost depends on the type of machine, region, and the cloud provider. The cost for IaaS or VMs for $n$ workers is given by

$$C_{IaaS} = \sum_{i=1}^{n} T_{vm\_i} * R_{vm\_i} \tag{3}$$

where $T_{vm\_i}$ is the time taken for processing and $R_{vm\_i}$ is the cost rate for the $i^{th}$ VM. Different cloud service providers offer different configurations of virtual machines and the cost rates vary for each cloud region. *Compute cost for Aggregator ($C_{FaaS}$):* Each invocation of the aggregator FaaS function is billed for the memory configured for the execution duration as follows:

$$C_{FaaS} = T_{FaaS} * M_{GB} * R_{mem} \tag{4}$$

where $T_{FaaS}$ is the execution time of the function, $M_{GB}$ is the memory configured for the function and $R_{mem}$ is the billing rate (vendor and geography specific) per memory-time consumed.

### 4.2 Storage Cost

Storage service is billed for the size of the data stored and the number of operations performed on the storage objects. These operations include writes, reads, deletes, lists, etc. Each operation has a unique billing rate.

Let $m$ be the model or gradient size to be stored in the storage service. Each worker writes a gradient file in the storage, and the aggregator writes the final updated model to storage. Hence the number of writes is $n + 1$, where $n$ is the number of workers. As mentioned in section 3, the aggregator gets triggered when a worker

writes local gradients to the storage. At each of these invocations of the aggregator, it lists all files present in storage to check whether all workers have written their gradients or not. Hence there are $n$ list operations in each batch. The aggregator function reads the gradient files from storage created by each of the workers. Similarly, each of the workers reads the final aggregated gradients from the storage resulting in $n * 2$ read operations. Each worker sends multiple read requests to the storage service till the aggregated gradients are saved by the aggregator and available for download. Although read requests failed till the aggregated gradients were available, these $l$ read requests made to storage incur charges. Hence, there are $(n * 2 + l)$ read requests billed. The aggregator deletes the $n$ model files created by the workers and only the updated model remains in the storage. Delete operations are free from all cloud providers. The total cost for storage per iteration is given as,

$$C_{Storage} = m * R_{st} + ((n * 2 + l) * R_{reads}) + ((n + 1) * R_{writes}) + (n * R_{lists}) \quad (5)$$

where $R_{st}$ is the cost rate per GB per month. $R_{reads}$, $R_{writes}$, and $R_{lists}$ are the rates of reads, writes, and lists respectively. Here Rate is for 1000 requests and *Rst* is the cost per GB per month.

### 4.3 Data transfer Cost

In a multi-cloud and multi-geography scenario, data is transferred across clouds and geographies of various CSPs. Generally, all incoming traffic or ingress is free for all cloud providers. However, all data transfers outside the cloud or geography are billed. The data transfer is billed for the total size of the data. Based on the cloud provider and geographies, either inter-geography or outward traffic rates are applied for the data transfers.

*Inter-cloud transfer:* When traffic leaves a particular cloud to other cloud providers or outside the internet, it is billed as per internet egress or outward transfer rates of the source cloud geography. Cost of Data transfer ($C_{DataTransfer}$) during an iteration of multi-cloud geo-distributed training consists of outward transfer from workers ($C_{DT\_IaaS}$), aggregator ($C_{DT\_FaaS}$) and storage ($C_{DT\_Storage}$).

$$C_{DataTransfer} = C_{DT\_IaaS} + C_{DT\_FaaS} + C_{DT\_Storage} \quad (6)$$

*Transfer from workers:* Each worker writes the gradients of size $m$ to the storage. The cost of data transfer for $n$ workers is

$$C_{DT\_IaaS} = \sum_{i=1}^{n} m * R_{vm\_i\_st} \quad (7)$$

where $R_{vm\_i\_st}$ is the cost rate of billing for transfer from the cloud region of $i^{th}$ VM to the storage cloud region.

*Transfer from FaaS:* Aggregator writes the updated model of size $m$ to the storage

$$C_{DT\_FaaS} = m * R_{agg\_st} \quad (8)$$

where $R_{agg\_st}$ is the cost rate of billing for transfer from the cloud region of the aggregator function to the storage cloud region.

*Transfer from Storage:* Aggregator reads the gradients from storage, one gradient file (size $m$) per worker. Total $n$ gradient files are transferred to the aggregator FaaS function. Additionally, each of the $n$ workers fetches the updated gradients of size $m$ from storage.

$$C_{DT\_Storage} = (n * m * R_{st\_agg}) + (\sum_{i=1}^{n} m * R_{st\_vm\_i}) \quad (9)$$

where $R_{st\_agg}$ is the cost rate of billing for transfer from the cloud region of storage to the aggregator cloud region. $R_{st\_vm\_i}$ is the cost rate of billing for transfer from the storage cloud region to the cloud region of $i^{th}$ VM.

## 5 EXPERIMENTAL SETUP

We conducted a comprehensive study on the GD training architecture, focusing on two use cases: our in-house recommender system (NISER) [11] using a Graph Neural Network Algorithm on the diginetica [8] dataset and sentiment analysis employing LSTM on the IMDB dataset. The recommender system is trained on 720K sessions and 43K product items, generating 17MB gradients, while sentiment analysis uses 50K movie reviews with 49MB gradients.

Models were trained in a distributed environment across Mumbai, London, Oregon, and Sydney. Workers ran on VMs from Amazon EC2, GCP, and Azure with consistent configurations (2 cores, 8GB memory). FaaS instances from AWS Lambda and GCP function deployed gradient aggregators configured with 1GB memory. Training data was fetched from AWS S3 storage, and gradients were stored on S3 and GCP storage. Completion time and cost for one mini-batch were recorded in all experiments, with cost calculated using CSP billing services.

## 6 EXPERIMENTAL ANALYSIS

In this section, we perform an analysis of experiments conducted by placing worker VMs and aggregators in various geographies and clouds.

### 6.1 Data transfer bandwidths

We study the bandwidth available when data is transferred from the worker VMs to a storage service. We consider 4 regions namely London, Mumbai, Oregon, and Sydney resulting in a total of 16 combinations of source and destination. VMs are from AWS, Azure and GCP. Storage services are from AWS and GCP. Hence for each source-destination chosen, there are 6 combinations of VM-storage service. Right side graph of Fig. 4 gives the bandwidth during an inter-geographies transfer, while left side graph shows the bandwidth when the source and destination are in the same geography.

- As expected, intra-region transfers have higher bandwidth as compared to inter-region bandwidths
- For inter-region, the transfer to AWS S3 from any of the three cloud worker VMs (AWS, Azure, GCP) has higher bandwidths than the transfer to GCP cloud storage (Fig. 4).
- For transfers within the same region, transfer to AWS S3 is better than transferring to GCP cloud storage in the majority of cases. The only exceptions are - GCP worker VM to GCP cloud storage transfer in London and Oregon regions (Fig. 4).

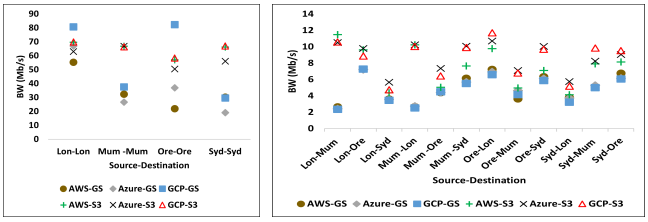



**Figure 4: Use case NISER: Upload bandwidth for gradient transfer from worker VM (AWS, Azure, GCP) in source region to Storage service (AWS S3, GCP Cloud storage) in a same and different geography.**

## 6.2 Effect of cloud vendor and aggregator on mini-batch time distribution

As previously mentioned, the total time for one mini-batch includes worker training time, gradient uploading, aggregation time, and aggregated gradient downloading. In this experiment, we investigate how aggregator placement and cloud vendor choice affect each aspect of the total mini-batch time. Specifically, we conduct experiments with all workers in GCP across four geographies: London, Sydney, Mumbai, and Oregon. We measure the mini-batch execution time using Google function and Google storage in each of these locations (5a). The same experiment is then repeated with VM workers in the AWS cloud across the same geographies, utilizing an AWS Lambda instance as an aggregator with AWS S3 storage (5b). We observe the following

- The maximum time in a mini-batch completion is consumed by the aggregator in waiting to receive gradients from all the workers. This is due to the synchronization of workers by the aggregator. This is followed by time spent in sending the gradient from the workers to the aggregator and back.
- Minimum time per mini-batch is consumed when worker VMs, aggregator, and associated storage reside in AWS cloud (Fig.5b) and the maximum time is consumed when worker VMs, aggregators, and storage are from GCP.
- While the total time to complete a mini-batch remains constant within a specific cloud, the duration of each phase varies based on aggregator placement. In Fig. 5a, although the total mini-batch time remains around 14 seconds regardless of the aggregator's region, there are significant variations in the time distribution across different stages when the aggregator's location is altered.

## 6.3 Effect of aggregator placement and cloud vendor

In this experiment, we study the effect of aggregator placement and the choice of cloud vendor on performance. In this set of experiments, we choose worker VMs for model training in multiple geographies but all from one cloud vendor at a time. However, the aggregator for each of these experiments is chosen from a combination of AWS Lambda, and Google functions with AWS S3 and GCP storage (GS). For example, Fig.6a shows data for worker VMs in AWS distributed in four different geographies. The latency and cost comparison is done by placing aggregator and storage combinations (Lambda+S3, Lambda+GS, GCP+S3, and GCP+GS) in Mumbai, London, Oregon, and Sydney. We observe the following:

- For both the use cases, we get minimum latency by placing VMs in the GCP cloud and using Lambda and S3 for aggregator and storage in Mumbai regions (Fig.6c and Fig.7c).
- For both the use cases, we get the minimum cost of deployment when placing all VMs in AWS and using Lambda and S3 for aggregator and storage respectively in the Oregon region (Fig. 6a and Fig. 7a).
- The choice of GCP function and GCP storage as aggregator results in higher latency in most of the cases irrespective of the cloud chosen for worker VMs.
- As expected, the use of worker VMs, aggregators, and storage from the same CSP is cost-efficient. However, the same is not true for the latency.

These observations can be attributed to the unique cost models and available network bandwidth between cloud services as well as cloud vendors.

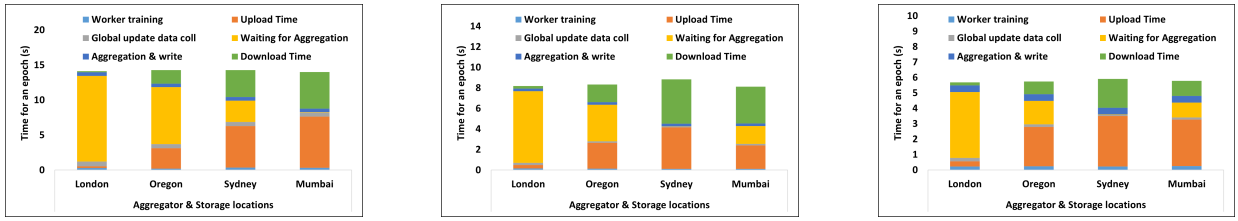
## 6.4 Effect of aggregator hierarchy

In this experiment, we analyze how the aggregator's placement impacts the cost and performance of training models for the NISER use case. Fig. 8 illustrates the cost and time required to complete one mini-batch when using global and regional aggregators with workers hosted in AWS and GCP clouds. It's evident that having a single global aggregator yields lower latency compared to an architecture employing regional aggregators in each geography alongside a global aggregator. This is mainly because, for a small number of workers, network latency overhead outweighs aggregate computation delays. Also, AWS Lambda instance as an aggregator results in a lower latency as compared to the GCP function instances. This is due to the higher compute capacity and network bandwidth observed in AWS as compared to GCP.

In case we use multiple regional aggregators (Fig. 8), the gradients are aggregated locally and transferred to storage. Hence, the total cost of gradient transfer is due to aggregator to storage transfer in each geography. Unlike the case of one global aggregator where all VMs in each geography transfer data to global storage. The difference between the overall cost for both of these deployments depends on participating geographies. For example, the cost of using only a global aggregator is 50% higher than having regional aggregators when the aggregator and storage are placed in London (Fig. 8) in AWS. This is due to the reason that gradients are coming to London aggregator and storage for aggregation from other geographies and VM egress cost is 5× and 4× higher in Sydney and Mumbai respectively compared to London [2].

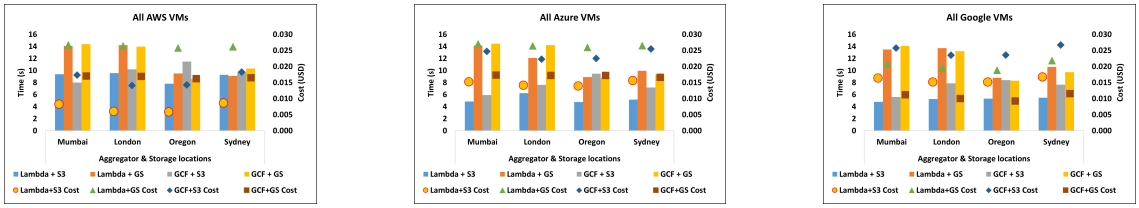
## 6.5 Cost Model Validation

To validate our cost model with use case NISER, we execute a multi-regional training experiment. This involves deploying 4 workers across distinct geographical regions utilizing AWS EC2 instances. S3 bucket is used as an intermediate storage with Lambda serving as an aggregator at one of the locations. Our experimentation encompassed 100 batches of training, during which we meticulously track costs from AWS cost management console. Furthermore, we



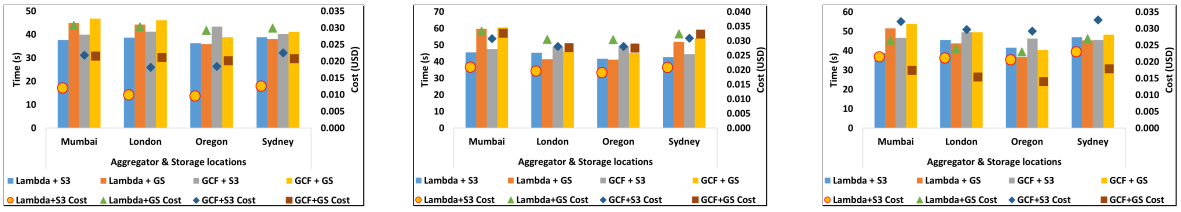

(a) (b) (c)

**Figure 5: Use case NISER: Time split for mini-batch execution with various deployment scenarios. (a) Worker VMs (GCP), Aggregator (GCP Functions) and Storage (GCP) (b) Worker VMs (AWS), Aggregator (AWS Lambda) and Storage (AWS S3) (c) Worker VMs (GCP), Aggregator (AWS Lambda) and Storage (AWS S3)**




(a) (b) (c)

**Figure 6: Use case NISER - One mini-batch completion time comparison when worker VMs run in (a) AWS only (b) Azure (c) GCP cloud. The aggregator (FaaS) and storage combinations are chosen from AWS and GCP and placed in one of the 4 geographies at a time in each of the three clouds**




(a) Only AWS VMs (b) Only Azure VMs (c) Only GCP VMs

**Figure 7: Use case Sentiment Analysis - One mini-batch completion time comparison when workers VMs run in (a) AWS only (b) Azure (c) GCP cloud. The aggregator (FaaS) and storage combinations are chosen from AWS and GCP and placed in one of the 4 geographies at a time in each of the three clouds**

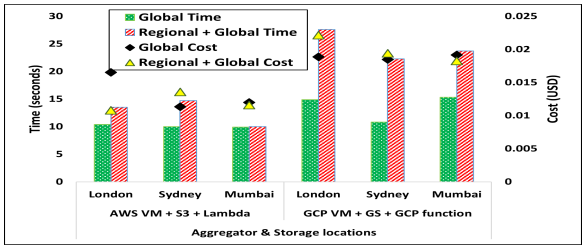



**Figure 8: Use case NISER: Regional Aggregator Effect in AWS and GCP**

projected costs for an identical experimental configuration using the cost model elucidated in section 4. Our cost model accurately forecasts costs with a marginal error of less than 3.5%.

## 7 CONCLUSION

In this work, we presented our study on geo-distributed training in a multi-cloud environment. We propose the use of serverless functions as gradient aggregators in conjunction with storage services from multiple CSPs. We study the performance of hierarchical aggregator architecture. Our experiments show that the choice of cloud vendor and placement of aggregators in geo-distributed training has a significant effect on the performance and cost of deployment. Additionally, we presented a cost model for estimating the cost of one mini-batch training in a multi-cloud environment. The proposed cost model predicts the cost of model training with a significant accuracy. We believe that the proposed cost model can be used for estimating the cost of a distributed training architecture in a multi-cloud environment.

# REFERENCES


[1] Syeda Nahida Akter and Muhammad Abdullah Adnan. 2020. WeightGrad: Geo-Distributed Data Analysis Using Quantization for Faster Convergence and Better Accuracy. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 546–556. https://doi.org/10.1145/3394486.3403097

[2] Amazon. [n. d.]. AWS Data Transfer. Accessed Sept. 30, 2023. https://docs.aws.amazon.com/cur/latest/userguide/cur-data-transfers-charges.html

[3] Amazon. [n. d.]. AWS Lambda. Accessed Apr. 12, 2023. https://aws.amazon.com/lambda/

[4] Amazon. [n. d.]. AWS S3. Accessed Apr. 12, 2023. https://aws.amazon.com/s3/

[5] Christopher Briggs, Zhong Fan, and Peter Andras. 2020. Federated learning with hierarchical clustering of local updates to improve training on non-IID data. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–9.

[6] Rafaela C Brum, Maria Clicia Stelling de Castro, Luciana Arantes, Lúcia Maria de A Drummond, and Pierre Sens. 2023. Multi-FedLS: a Framework for Cross-Silo Federated Learning Applications on Multi-Cloud Environments. *arXiv preprint arXiv:2308.08967* (2023).

[7] Leandro Costa da Silva, Robson De Medeiros, and Nelson Rosa. 2023. COSTA: A Cost-Driven Solution for Migrating Applications in Multi-Cloud Environments. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing* (Tallinn, Estonia) *(SAC '23)*. Association for Computing Machinery, New York, NY, USA, 57–63. https://doi.org/10.1145/3555776.3577718

[8] GitHub. [n. d.]. Diginetica Dataset. Accessed Sept. 29, 2023. https://darel13712.github.io/rs_datasets/Datasets/diginetica/

[9] Google. [n. d.]. Google Cloud Storage. Accessed Apr. 12, 2023. https://cloud.google.com/storage

[10] Google. [n. d.]. Google Functions. Accessed Apr. 12, 2023. https://cloud.google.com/functions

[11] Priyanka Gupta, Diksha Garg, Pankaj Malhotra, Lovekesh Vig, and Gautam M Shroff. 2019. NISER: normalized item and session representations with graph neural networks. *arXiv preprint arXiv:1909.04276* (2019).

[12] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 629–647. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/hsieh

[13] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. 2017. Gaia:{Geo-Distributed} machine learning approaching {LAN} speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 629–647.

[14] Jiawei Jiang, Shaoduo Gan, Yue Liu, Fanlin Wang, Gustavo Alonso, Ana Klimovic, Ankit Singla, Wentao Wu, and Ce Zhang. 2021. Towards Demystifying Serverless Machine Learning Training. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 857–871. https://doi.org/10.1145/3448016.3459240

[15] Mashal Khan, Frank G. Glavin, and Matthias Nickles. 2023. Federated Learning as a Privacy Solution - An Overview. *Procedia Computer Science* 217 (2023), 316–325. https://doi.org/10.1016/j.procs.2022.12.227 4th International Conference on Industry 4.0 and Smart Manufacturing.

[16] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 583–598. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu

[17] Yongyao Li, Chenyu Fan, Xiaoning Zhang, and Yufeng Chen. 2023. Placement of parameter server in wide area network topology for geo-distributed machine learning. *Journal of Communications and Networks* (2023).

[18] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William Dally. 2018. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. https://openreview.net/pdf?id=SkhQHMW0W

[19] Lumin Liu, Jun Zhang, S. H. Song, and Khaled B. Letaief. 2020. Client-Edge-Cloud Hierarchical Federated Learning. In *2020 IEEE International Conference on Communications, ICC 2020 - Proceedings*. https://doi.org/10.1109/ICC40277.2020.9148862

[20] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M Hellerstein. 2012. Distributed graphlab: A framework for machine learning in the cloud. *arXiv preprint arXiv:1204.6078* (2012).

[21] Yucheng Low, Joseph E Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E Guestrin, and Joseph Hellerstein. 2014. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041* (2014).

[22] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 54)*, Aarti Singh and Jerry Zhu (Eds.). PMLR, 1273–1282. https://proceedings.mlr.press/v54/mcmahan17a.html

[23] Microsoft. [n. d.]. Azure Functions,. Accessed Apr. 12, 2023. https://azure.microsoft.com/en-in/services/functions/

[24] Chetan Phalak, Dheeraj Chahal, Manju Ramesh, and Rekha Singhal. 2023. mSIRM: Cost-Efficient and SLO-aware ML Load Balancing on Fog and Multi-Cloud Network. In *Proceedings of the 13th Workshop on AI and Scientific Computing at Scale using Flexible Computing*. 19–26.

[25] Chetan Phalak, Dheeraj Chahal, and Rekha Singhal. 2023. SIRM: Cost efficient and SLO aware ML prediction on Fog-Cloud Network. In *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 825–829.

[26] Manju Ramesh, Dheeraj Chahal, and Rekha Singhal. 2023. Multicloud Deployment of AI Workflows Using FaaS and Storage Services. In *2023 15th International Conference on COMmunication Systems NETworkS (COMSNETS)*. 269–277. https://doi.org/10.1109/COMSNETS56262.2023.10041365

[27] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems 24*, Vol. 24. 693–701.

[28] Pankaj Sahu, Shubhro Roy, Mangesh Gharote, Sutapa Mondal, and Sachin Lodha. 2022. Cloud Storage and Processing Service Selection considering Tiered Pricing and Data Regulations. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. 92–101.

[29] Mehdi Salehi Heydar Abad, E. Ozfatura, Deniz Gündüz, and Ozgur Ercetin. 2020. Hierarchical Federated Learning ACROSS Heterogeneous Cellular Networks. 8866–8870. https://doi.org/10.1109/ICASSP40776.2020.9054634

[30] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *CoRR* abs/1802.05799 (2018). arXiv:1802.05799 http://arxiv.org/abs/1802.05799

[31] Shaohuai Shi, Xianhao Zhou, Shutao Song, Xingyao Wang, Zilin Zhu, Xue Huang, Xinan Jiang, Feihu Zhou, Zhenyu Guo, Liqiang Xie, et al. 2021. Towards scalable distributed training of deep learning on public cloud clusters. *Proceedings of Machine Learning and Systems* 3 (2021), 401–412.

[32] Wenting Tan, Xiao Shi1, Cunchi Lv, and Xiaofang Zhao. 2023. Cloudless-Training: A Framework to Improve Efficiency of Geo-Distributed ML Training. arXiv:2303.05330 [cs.DC]

[33] Paul Voigt and Axel von dem Bussche. 2017. *The EU General Data Protection Regulation (GDPR): A Practical Guide* (1st ed.). Springer Publishing Company, Incorporated.

[34] Yang Xiang, Zhihua Wu, Weibao Gong, Siyu Ding, Xianjie Mo, Yuang Liu, Shuohuan Wang, Peng Liu, Yongshuai Hou, Long Li, et al. 2022. Nebula-I: A general framework for collaboratively training deep learning models on low-bandwidth cloud clusters. *arXiv preprint arXiv:2205.09470* (2022).

[35] Jie Zhu, Shenggui Li, and Yang You. 2022. Sky Computing: Accelerating Geo-distributed Computing in Federated Learning. *arXiv preprint arXiv:2202.11836* (2022).

# Hypergraphs: Facilitating High-Order Modeling of the Computing Continuum

Dragi Kimovski
dragi.kimovski@aau.at
University of Klagenfurt
Klagenfurt, Austria

## ABSTRACT

As contemporary computing infrastructures evolve to include diverse architectures beyond traditional von Neumann models, the limitations of classical graph-based infrastructure and application modelling become apparent, particularly in the context of the computing continuum and its interactions with Internet of Things (IoT) applications.

Hypergraphs prove instrumental in overcoming this obstacle by enabling the representation of computing resources and data sources irrespective of scale. This allows the identification of new relationships and hidden properties, supporting the creation of a federated, sustainable, cognitive computing continuum with shared intelligence.

The paper introduces the HyperContinuum conceptual platform, which provides resource and applications management algorithms for distributed applications in conjunction with next-generation computing continuum infrastructures based on novel von Neumann computer architectures. The HyperContinuum platform outlines high-order hypergraph applications representation, sustainability optimization for von Neumann architectures, automated cognition through federated learning for IoT application execution, and adaptive computing continuum resources provisioning.

## KEYWORDS

Hypergraphs, Computing Continuum, Optimisation

**ACM Reference Format:**
Dragi Kimovski. 2024. Hypergraphs: Facilitating High-Order Modeling of the Computing Continuum. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3629527.3651423

## 1 INTRODUCTION

The digital representation of the physical universe is a complex task that requires understanding the concept of morphism first. Morphism is a way to describe how different parts of a shape or structure relate to each other mathematically. It was first introduced by the French mathematician Henri Poincare in 1895 [1]. More than seventy years later, in 1968, the American mathematician Haskell Curry and logician William Alvin Howard applied this concept to computer science. They described the Curry-Howard correspondence, which shows that the proof of a computer system and the model of computation are the same kind of mathematical object [2]. This means we can model computer programs and systems as directed graphs commonly used today to represent infrastructures, data, and applications.

However, graphs have limitations when it comes to modelling modern *computing continuum* infrastructures and their interactions with Internet of Things (IoT) applications involving millions of data sources, as they can not express the scale of the data sources or application instances.

The problem of using graphs for modeling of distributed systems is further aggravated as researchers have recently integrated novel computing architectures in the computing continuum beyond the ones based on the traditional stored program von Neumann model, where the programs and data are stored in a single operating memory [3]. These novel architectures use different production processes, processing implementation, data representations and distributed memory models, known as non-Von Neumann architectures. They range from power-efficient single-board Artificial Intelligence (AI) accelerators to Quantum and Neuromorphic computers [4, 5]. While these architectures hold great potential for revolutionizing data processing and analysis in healthcare, transportation, and entertainment, integrating them with established cloud and edge computing paradigms remains challenging due to significant architectural heterogeneity, data representation, communication, and limited modelling tools. Unfortunately, extending the computing continuum with non-Von Neumann architectures causes multiple difficulties in application and infrastructure modelling, resource provisioning and execution optimisation [6].

To address the complexity of the computing continuum and the integration of non-Von Neumann architectures, we discuss the concept of hypergraphs, powerful mathematical objects generalising graphs [7], as potent tools for modelling. In hypergraphs, hyperedges can connect any number of hypervertices. Hypergraphs are more expressive than pair-wise classical graphs, allowing us to model the computing continuum and extreme-scale applications as mathematical objects with higher-order, high-dimensional relations. They can represent computing resources and data sources, regardless of their scale. Therefore, we can identify new relationships between resources and data sources by abstracting from the scale. For example, we can use a hypervertex to represent a set of computing continuum resources and connect it with a hyperedge to another hypervertex representing various data sources, regardless of their scale. By leveraging hypergraphs, we can expose previously unknown relations between the resources and identify hidden properties of the applications. To illustrate the benefits of

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3651423

these modelling approaches, let us discuss an example, in which a distributed application composed of three components interconnected in a specific topology is deterministically deployed on a given computing continuum infrastructure, which, in continuation, is connected to multiple sensing devices that monitor the environment. This implies that for modelling the system using classical graphs, we have to consider the application, infrastructure and sensor graphs in isolation and only afterwards to identify their interactions manually. On the other hand, we can model the application, infrastructure, sensing devices, and environment using hypergraphs as hyperedges. The hypervertices model the interactions between these hyperedges. This allows us to move away from the fixed size of the application and infrastructure and models them nevertheless of the scale (how many components, instances, infrastructures and sensing devices are available).

Therefore, the paper proposes the HyperContinuum conceptual platform for sustainable and scalable distributed applications processing over computing continuum infrastructures based on a hypergraph (HG) representation of the data, environment, infrastructure, and applications.

The paper has four sections. We first survey the related work in Section 2. Afterwards, we present the proposed conceptual architecture in Section 3 and Section 4 concludes the paper.

## 2 RELATED WORK

This section details the state-of-the-art optimisation of container orchestration systems.

### 2.1 Hypergraph applications modelling and sustainability analysis

The current application analysis approaches rely on pair-wise ordinary graphs or state machine representations to model the applications and the infrastructure below [7]. This limits their application for highly adaptive and heterogeneous systems, such as the computing continuum. From a performance point of view, the classical workflow and hardware-specific optimization approaches brought significant improvements in distributed applications execution, specifically in performance, energy management, and financial cost [8]. Unfortunately, these approaches primarily support large data centers with a relatively homogeneous set of resources with static topologies and performance profiles. They lack functionality for supporting complex application workflows, which can change the structure and conditional execution branches based on the input parameters and workload. Concretely, ordinary workflows and dynamics (i.e., changing the number and content of vertices and edges) lead to high variability in computational needs [9]. Therefore, the hypergraph and hyperworkflow models can be intelligently transformed into ordinary workflows for improved performance prediction. They demonstrate they can enable conditional algorithm/execution branch selection and advanced auto-scaling techniques to ensure better performance. Furthermore, energy consumption is a primary component of a computing infrastructure's total cost of ownership. Power consumption and thermal dissipation limit the achievable peak performance with lower cost.

### 2.2 Hyperworkflow optimisation and cognition with federated learning

Federated learning techniques where multiple decentralised devices or nodes collaborate to train a shared model while keeping data local to the devices [10]. It can be used to optimise hyperworkflows by improving the accuracy and efficiency of the model while ensuring the privacy and security of data. In hyperworkflows optimization, federated learning can be applied in several ways. One approach is to use federated learning to optimise decision-making processes in hyperworkflows, such as determining the next best task or predicting execution outcomes. Another approach is to use federated learning to improve the performance of machine learning models used in workflows. State-of-the-art methods in federated learning for workflow optimization involve advanced techniques such as federated transfer learning and federated reinforcement learning. These methods address challenges such as communication overhead, data heterogeneity, and model convergence in federated learning systems. Existing workflow management systems such as Pegasus [11], Apollo [12], and Askalon are centralised systems and either do not support machine learning (ML)-based workflow execution optimization or utilise conventional and centralized ML approaches. In such systems, ML systems are centralised, and workers periodically send local updates about the workload to a set of parameter servers, such as Tensorflow and traditional federated learning systems [13].

### 2.3 Overlay infrastructure provisioning

The workloads in the distributed computing continuum are often machine learning and data-intensive and have high requirements for performance; deployment strategies, e.g., offloading, computing close to data, and parallelizing the heavy tasks, are required due to the constraints of capacity, time constraints, and energy [14]. Therefore, infrastructure provisioning and deployment planning algorithms have been proposed based on critical paths, graph decomposition, and potential data traffic [15]. Most of the early work is based on data workflows with a deterministic performance model of the components on a set of given computing nodes. The data processing or machine learning workloads heavily depend on the volume and availability of the data, which results in new challenges to apply those existing approaches [16]. Machine learning-based infrastructure provisioning and deployment management, e.g., reinforcement learning, has been proposed in the past years; however, those approaches face challenges of low robustness when the workload patterns change.

## 3 CONCEPTUAL ARCHITECTURE

This section presents a conceptual architecture of the HyperContinuum framework.

The HyperContinuum high-level framework involves two conceptual layers displayed in Figure 1, creating an automated, sustainable loop for managing distributed applications as hypergraphs over the computing continuum with non-Von Neumann architectures:

- Hypergraph cognition layer covering the creation, optimisation and analysis of the hypergraphs;

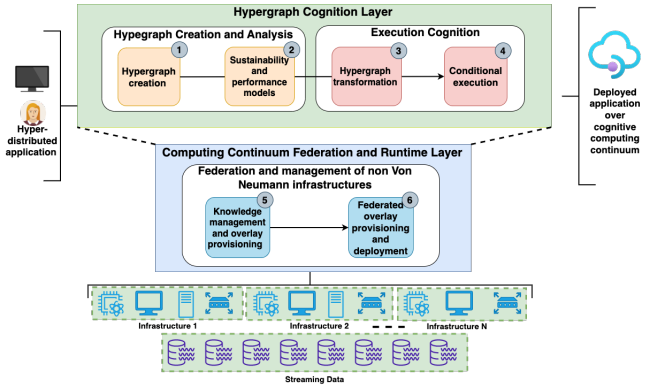



**Figure 1: Conceptual life cycle of the HyperContinuum framework.**

- Federation and management of non-Von Neumann infrastructures layer that focuses on provisioning computing continuum resources as logical overlays.

The *hypergraph cognition layer* facilitates creating, analysing, and optimising the applications' hyperworkflows comprising performance and energy models and optimization techniques. Hypergraph creation with sustainability and performance analysis analyses the distributed application's higher-order interactions with the environment and infrastructure and represents it as the hypergraph (See Figure 2a and b). Unlike classical approaches for managing distributed applications, which model the applications in isolation from the infrastructure and environment, we generalize the applications and the infrastructure as hypergraphs [17]. The hypergraphs allow us to model the distributed application components and all multi-dimensional interactions, including the interactions with the environment. This enables us to use a more realistic representation of the application and its interaction with the environment. Thereafter, as depicted in Figure 2c, when the data patterns and directions of the interactions between the application and environment are known, the hypergraph is transformed into a directed acyclic hypergraph (DAH). During the transformation from HG to DAH multiple sustainability metrics are considered, including energy wastage estimation by creating benchmarks for energy and performance modelling of non-Von Neumann architectures. The execution cognition, including the hypergraph transformation and conditional execution, utilises an intelligent distributed approach for cognitive optimization of the application DAH considering the possibility for conditional execution of the application branches based on external factors, such as users' location and cached results [18]. Furthermore, it applies an intelligent data distribution algorithm to only store the information of the applications and users on trusted storage infrastructures, thus complying with the data security standards [19]. Based on the workload, environmental parameters, input data, and scale of the systems and application, the system can transform the relevant part of the DAH to a directed acyclic graph (DAG) specifically tailored for the given execution depicted in Figure 2d.

The computing continuum *federation and management of non-Von Neumann infrastructures* layer focuses on the execution aspect

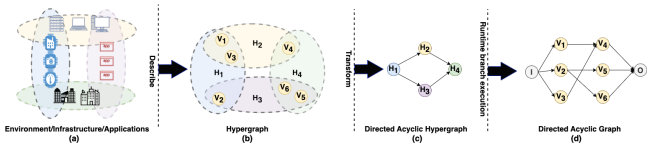



**Figure 2: Transformation from hypergraph to ordinary graph.**

of the DAH by providing automated configuration and provisioning of interoperable infrastructures and deployment. Federated infrastructure knowledge management and overlay provisioning enable provisioning over multiple computing continuum systems. It enables the creation of an interoperable resources overlay that includes heterogeneous hardware resources, including non-Von Neumann hardware, across multiple computing continuum infrastructures by implementing novel infrastructure knowledge management algorithms and approaches [20]. In addition, the layer manages the deployment of the given branches of the DAH as DAG, making it interoperable with any existing system. Lastly, it identifies suitable storage sites to create a virtualized distributed data federation and provides continuous infrastructure and later application monitoring [21].

## 4 CONCLUSION

This paper introduces a novel conceptual framework for modelling of the computing continuum and IoT applications as hypergraphs, with a primary focus on non-Von Neumann systems. The limitations of classical graph-based modeling in accommodating diverse architectures beyond traditional von Neumann models are highlighted. The utilization of hypergraphs emerges as a crucial solution, allowing for the representation of computing resources and data sources at any scale. This innovation facilitates the identification of new relationships and hidden properties, laying the foundation for the development of a federated and sustainable computing continuum.

The HyperContinuum conceptual platform describes novel concepts for resource and applications management algorithms tailored for distributed applications within next-generation computing continuum infrastructures based on novel von Neumann computer architectures. The platform introduces the concepts for high-order hypergraph applications representation, sustainability optimization for von Neumann architectures, automated cognition through federated learning for IoT application execution, and adaptive computing continuum resources provisioning. Overall, the HyperContinuum platform not only discusses the challenges posed by diverse computing infrastructures but also sets the stage for the future development of intelligent and sustainable systems. The paper underscores the importance of embracing hypergraph-based models in shaping the next era of computing, marking a significant step towards creating a more efficient and interconnected computing environment.

## REFERENCES

[1] Jiří Adámek, Horst Herrlich, and George Strecker. *Abstract and concrete categories.* Wiley-Interscience, 1990.

[2] C-HL Ong and Charles A Stewart. A curry-howard foundation for functional computation with control. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 215–227, 1997.

[3] Antara Ganguly, Rajeev Muralidhar, and Virendra Singh. Towards energy efficient non-von neumann architectures for deep learning. In *20th international symposium on quality electronic design (ISQED)*, pages 335–342. IEEE, 2019.

[4] National Academies of Sciences. Quantum computing: progress and prospects. 2019.

[5] Dragi Kimovski, Roland Mathá, Josef Hammer, Narges Mehran, Hermann Hellwagner, and Radu Prodan. Cloud, fog, or edge: Where to compute? *IEEE Internet Computing*, 25(4):30–36, 2021.

[6] Reza Farahani, Dragi Kimovski, Sashko Ristov, Alexandru Iosup, and Radu Prodan. Towards sustainable serverless processing of massive graphs on the computing continuum. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, pages 221–226, 2023.

[7] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete applied mathematics*, 42(2-3):177–201, 1993.

[8] Vincenzo De Maio and Dragi Kimovski. Multi-objective scheduling of extreme data scientific workflows in fog. *Future Generation Computer Systems*, 106:171–184, 2020.

[9] Björn B Brandenburg, John M Calandrino, and James H Anderson. On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In *2008 Real-Time Systems Symposium*, pages 157–169. IEEE, 2008.

[10] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of machine learning and systems*, 1:374–388, 2019.

[11] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[12] Fedor Smirnov, Behnaz Pourmohseni, and Thomas Fahringer. Apollo: Modular and distributed runtime system for serverless function compositions on cloud, edge, and iot resources. In *Proceedings of the 1st Workshop on High Performance Serverless Computing*, pages 5–8, 2020.

[13] Guillem Ramirez-Gargallo, Marta Garcia-Gasulla, and Filippo Mantovani. Tensorflow on state-of-the-art hpc clusters: a machine learning use case. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 526–533. IEEE, 2019.

[14] Dumitru Roman, Radu Prodan, Nikolay Nikolov, Ahmet Soylu, Mihhail Matskin, Andrea Marrella, Dragi Kimovski, Brian Elvesæter, Anthony Simonet-Boulogne, Giannis Ledakis, et al. Big data pipelines on the computing continuum: Tapping the dark data. *Computer*, 55(11):74–84, 2022.

[15] Thang Le Duc, Rafael García Leiva, Paolo Casari, and Per-Olov Östberg. Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. *ACM Computing Surveys (CSUR)*, 52(5):1–39, 2019.

[16] Ali Shahidinejad and Mostafa Ghobaei-Arani. Joint computation offloading and resource provisioning for e dge-cloud computing environment: A machine learning-based approach. *Software: Practice and Experience*, 50(12):2212–2230, 2020.

[17] João Paulo A Almeida. Model-driven design of distributed applications. In *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops: OTM Confederated International Workshops and Posters, GADA, JTRES, MIOS, WORM, WOSE, PhDS, and INTEROP 2004, Agia Napa, Cyprus, October 25-29, 2004. Proceedings*, pages 854–865. Springer, 2004.

[18] José Carlos Fonseca, Vincent Nélis, Gurulingesh Raravi, and Luís Miguel Pinho. A multi-dag model for real-time parallel applications with conditional execution. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1925–1932, 2015.

[19] Lori M Kaufman. Data security in the world of cloud computing. *IEEE Security & Privacy*, 7(4):61–64, 2009.

[20] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Adaptive resource configuration for cloud infrastructure management. *Future Generation Computer Systems*, 29(2):472–487, 2013.

[21] Polona Štefanič and Vlado Stankovski. Multi-criteria decision-making approach for container-based cloud applications: the switch and entice workbenches. *Tehnički vjesnik*, 27(3):1006–1013, 2020.

# Resource Demand Profiling of Monolithic Workflows

Ivo Rohwer
ivo.rohwer@uni-wuerzburg.de
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

Maximilian Schwinger
maximilian.schwinger@dlr.de
German Aerospace Center (DLR)
Oberpfaffenhofen, Germany

Nikolas Herbst
nikolas.herbst@uni-wuerzburg.de
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

Peter Friedl
peter.friedl@dlr.de
German Aerospace Center (DLR)
Oberpfaffenhofen, Germany

Michael Stephan
michael.stephan@lrz.de
Leibniz Rechenzentrum
Garching, Germany

Samuel Kounev
samuel.kounev@uni-wuerzburg.de
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

## ABSTRACT

We propose a novel approach for resource demand profiling of resource-intensive monolithic workflows that consist of different phases. Workflow profiling aims to estimate the resource demands of workflows. Such estimates are important for workflow scheduling in data centers and enable the efficient use of available resources. Our approach considers the workflows as black boxes, in other words, our approach can fully rely on recorded system-level metrics, which is the standard scenario from the perspective of data center operators. Our approach first performs an offline analysis of a dataset of resource consumption values of different runs of a considered workflow. For this analysis, we apply the time series segmentation algorithm PELT and the clustering algorithm DBSCAN. This analysis extracts individual phases and the respective resource demands. We then use the results of this analysis to train a Hidden Markov Model in a supervised manner for online phase detection. Furthermore, we provide a method to update the resource demand profiles at run-time of the workflows based on this phase detection. We test our approach on Earth Observation workflows that process satellite data. The results imply that our approach already works in some common scenarios. On the other hand, for cases where the behavior of individual phases is changed too much by contention, we identify room and next steps for improvements.

## CCS CONCEPTS

**• Software and its engineering → Process management**; **• General and reference → Cross-computing tools and techniques**.

## KEYWORDS

Phase Detection, HMM, Workflow Profiling

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE Companion '24 , May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651425

**ACM Reference Format:**
Ivo Rohwer, Maximilian Schwinger, Nikolas Herbst, Peter Friedl, Michael Stephan, and Samuel Kounev. 2024. Resource Demand Profiling of Monolithic Workflows. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '24 ), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3629527.3651425

## 1 INTRODUCTION

As the possibilities for collecting data increase year on year, the efficient processing of these data volumes is becoming an ever-growing challenge. A frequently used approach for processing large amounts of data are scientific workflows. Data centers are trying to make the best possible use of their available resources without causing a decline in the quality of service. To achieve this, the prediction of resource demands plays an important role, as these estimates are the basis for workflow scheduling decisions. The simplest but often used resource demand estimates focus on determining a worst-case value for each workflow and do not take into account the occurrence of different phases with different resource demands within a workflow. On the other hand, some approaches use the measured resource utilization history to predict the resource utilization for a future time period of a specified fixed length. Even more precise predictions are possible using approaches that predict the resource demand of individual workflows taking into account different phases. These approaches use unsupervised clustering methods, such as k-means or Hidden Markov Models (HMMs). In this paper, we present a new approach that enables supervised training of a HMM by combining offline and online algorithms. In this way, as few as just recorded systems-level metrics of about 30 executions are required as training data. Furthermore, we provide a method that uses the online phase detection of the HMM to update online the offline-created resource demand profiles.

The use of a method that predicts the resource demand of a single workflow makes sense for monolithic workflows that have such a high resource consumption that it is worth predicting each individual workflow. We select as application domain Earth Observation (EO) workflows. EO workflows are scientific workflows that process satellite data and, for example, are used to determine the condition of forests [11], record the movement of Antarctic ice [2], or monitor daily changes in global freshwater bodies [7]. Because of the mentioned characteristics of EO workflows, we use them to

**Resource Demand Profiles**

We understand a resource demand profile to be an estimate of the resource demands of a specific workflow over time. The profile indicates for the workflow under consideration how long it is expected to run at maximum and indicates for each point in time during this period how high the maximum resource demand is expected to be.

evaluate our approach. We consider the workflows as black boxes, but we assume that we have a data set of resource consumption values available for each individual workflow.

The remainder of this workshop paper is organized as follows: In Section 2, we summarize related work. In Section 3, we describe our approach and discuss the algorithms we chose for the individual steps of our approach. In addition, in Section 4, we show the performed experiments and present the results. Finally, in Section 5, we give a conclusion and discuss future work.

## 2 RELATED WORK

In the following, we present briefly related work regarding our approach. **Workflow prediction for sets of applications** There are many workflow prediction approaches that use classic time series prediction methods for resource demand prediction, like for example, [10], [14] or [13]. This is not an ideal solution in the scenario we consider, as most time series prediction approaches are not designed to predict changepoints of time series, but rather, like ARIMA or ARMA for example, assume that the time series retains its statistical properties, which is precisely not the case after a phase transition that can be seen as a change point of the resource consumption time series. Different papers have also been published in this area using HMMs, such as the work by Kahn et al. [6]. They do take phases into account, but only to the extent that they determine the current phase and predict the resource demands of the next phase: The approach deals only with intervals of fixed length (for example 15-minute intervals in their paper). In addition, their approach assigns a phase only one of five possible CPU utilization levels. In contrast, we model sequences of phases of arbitrary lengths and resource demands. Furthermore, we adjust flexibly our resource demand profiles based on the observed phase lengths of already completed phases. Another example of the use of HMMs for workflow prediction is the work of Adel et al. [1]. In their approach, a HMM is trained for each resource, which is then used to predict the utilization of the individual resource, whereby they distinguish between four different load state classes. The goal of their approach was to use the predictions for autoscaling. The approach was tested successfully in a simulation.

**Resource demand prediction of individual applications** Gupta et al. [5] proposed a relatively simple approach to phase-based resource demand estimation based on the k-means algorithm. This is used to cluster the time series into phases. Subsequently, a phase transition table can be created based on these results. During operation, the k-means algorithm is also used to identify the current phase and then the upcoming phases are predicted using the transition table. Furthermore, in this approach, a table is created to track the CPU utilization of phase combinations of different workflows in order to make scheduling decisions based on this information.

Prats et al. [8] proposed an approach for phase-based prediction of workflows using HMM to profile individual workloads. However, unlike us, they train their HMM unsupervised, i.e., they also use it to extract the phases from the time series of resource consumption values. In contrast to this, we use deterministic algorithms to extract the phases from these time series. With the labels created in this way, we train the HMM in a supervised manner and use it for phase detection and prediction.

This makes it possible to predict the phase progression of a workflow from start to finish. This is not possible when using the unsupervised approaches, as it is not possible to prevent two separate segments in a run from being assigned to the same phase type. However, this means that no start-to-finish predictions can be derived from the phase transition tables resulting from these segmentations, as circles are possible in the phase sequences.

## 3 APPROACH

As described in the introduction, the goal of our work is to create a model for the resource demand estimation of monolithic workflows that we regard as black boxes. We assume that a dataset of recorded monitoring data from different example runs for a considered workflow type is available.

The first step is to divide the example runs from the data-set into meaningful phases. This is done in our approach by the PELT algorithm. Once the example runs are segmented into meaningful phases, we have to determine which of these segments from different example runs belong to the same phase. It is not possible to derive this information from the order of the phase segments, since, for example, some phases can only occur optionally or can be changed in their properties by contention. For this reason, we use the DBSCAN clustering algorithm for this task. Each of the resulting clusters represents one phase type. All identified phase types can now be characterized in terms of their run durations and resource demands. With this information, it is possible to estimate the possible start and end dates of the individual phases. Based on these estimates, a resource demand profile can be created for a workflow, which gives at each point in time a maximum resource demand value of all phases that could possibly be active at that point in time.

Our approach aims to update the estimates of the start times of the individual phases at the runtime of the individual workflow instances. This makes it possible to significantly increase the accuracy of the profiles: For example, if a phase of a workflow has a high resource demand, the appropriate amount of resources must be reserved for this phase for the entire time in which it may occur. This time is usually significantly longer than the actual maximum runtime of this phase, as we do not know exactly when this phase actually will start. This means that in this case, the actual resource demands of a workflow are significantly overestimated by the initial resource demand profile. However, online phase recognition makes it possible to narrow down the estimates of the start times of the remaining phases. With this information, the resource demand profile can then be updated and the overestimation of resource demands can be significantly reduced. For the task of online phase detection,

we train a HMM in a supervised manner, based on this phase segmentation and labeling. Our code and data artifacts are available for reproducibility and reusability via a CodeOcean Capsule at [9].

### 3.1 Resource Demand Profile Creation

Our approach creates a resource demand profile at the start of a workflow as well as at each recognized phase transition. The first step of our resource demand profiling approach is to determine all possible phase sequences that may follow the current phase. This is done by evaluating the transition probability table of the HMM.

Each of these possible phase sequences is then considered individually. The minimum possible start time and the maximum possible end time are calculated for each phase in each sequence. We assume the minimum duration of a phase to be the mean duration minus two standard deviations and the maximum length to be the mean duration plus two standard deviations. The earliest and the latest point in time at which a phase can possibly start in a considered sequence is given by the sum of either the minimum or the maximum lengths of all previous phases of this phase. Now, it is possible to calculate for each phase sequence for any time step all phases that can possibly occur at this time step. For each time step, we can now calculate the maximum of all mean resource consumption values plus two standard deviations of all phases that may occur in this time step.

If these maximum values are calculated for all time steps of all possible phase sequences that can follow from the current phase, the final resource demand profile can be calculated as follows: For each time step, we iterate through all values at this time step of the different sequences and take the maximum value for this time step. In this way, we can create an upper-bound estimation for the future demand of a specific resource for the considered workflow.

## 4 PRELIMINARY EVALUATION

We evaluate our approach on the terrabyte platform [3, 4] hosted at the Leibniz Rechenzentrum. As described in the introduction, we profile EO workflows to test our approach. In the following, we present the results of our experiments with the Multi-SAR workflow [12] regarding the memory consumption profile of this workflow. The Multi-SAR workflow processes data from radar satellites. We consider a scenario where a set of Multi-SAR workflow instances is to be executed concerning a certain memory limit. We compare the results of ASAP (as soon as possible) scheduling of these workflow instances in combination with different resource demand profile types including our approach. We measure the total execution times for the entire workflow set, i.e. the time span between the start of the first workflow instance and the end of the last workflow instance. The following three profile types are compared by us:

(1) **Constant** resource demand profiles, which contain a value based on the maximum resource consumption of a workflow
(2) **Static** resource demand profiles, which are non-constant and take into account the different resource demands of individual phases but are not updated at the runtime of the workflows
(3) **Dynamic** resource demand profiles, which are created by our approach and are updated at the runtime of individual workflow instances

The static profiles correspond to the initial profiles generated by our model but are not updated at the runtime of the workflows.

In Table 1, the exact number of time steps required to complete 20 instances of the Multi-SAR workflow that were scheduled by ASAP in combination with the three compared resource demand profile types. As we can see, the time required in the case of constants and static profiles does not differ significantly in the scenario under consideration. In contrast, our approach seems to significantly speed up the execution of the workflows: In this experiment, the workflows were completed more than 23% faster without exceeding the resource limit.

| *limit* | *number of workflow instances* | *profiling approach* | *duration in time steps* | *improvement vs constant* |
|---|---|---|---|---|
| 200 GB | 20 | constant | 650 | - |
| 200 GB | 20 | static | 641 | 1.38% |
| 200 GB | 20 | dynamic | 499 | 23.23% |

**Table 1: The table shows the results of our experiments on the terrabyte platform in terms of the time required for 20 workflow instances. The improvement is shown in comparison with the constant procedure.**

The results of our experiments show that our approach works for the memory consumption values of the Multi-SAR workflow. However, additional experiments have shown that our approach runs into problems if the behavior of individual phases is changed too much by contention. This is especially the case with time series of CPU data from multi-threaded workflows, as these time series exhibit a high variance.

## 5 CONCLUSION AND FUTURE WORK

In this paper, we present an approach for phase-based workflow profiling and online phase-detection for monolithic workflows. We use offline time series segmentation and density-based clustering to identify individual phases in a resource consumption value time series dataset. Subsequently, we train a HMM for online phase detection of the corresponding workflow. Furthermore, we present a method to utilize this phase detection to adjust the generated resource demand profiles at the runtime of the workflows. We evaluate our approach using Earth Observation workflows. Our experiments regarding memory consumption show that our approach works well in scenarios where the behavior of individual phases is not changed too much by contention.

However, if contention changes the behavior of individual phases too much, our approach has problems creating a consistent phase model. Here it seems to be a problem to call the PELT algorithm with the same hyper-parameters for highly different time series. Therefore, we aim to optimize our approach in this respect. In addition, further experiments are also needed, especially experiments that combine our approach with more specialized scheduling algorithms, in order to evaluate how well our approach would work in practice.

## ACKNOWLEDGMENTS

This research is funded by the Bavarian Research Institute for Digital Transformation (bidt), an institute of the Bavarian Academy of Sciences and Humanities as part of the project ROOT: Real-time earth Observation of fOrest dynamics and biodiversiTy (KON-22-024).

## REFERENCES

[1] Ahmed Adel and Amr El Mougy. 2022. Cloud Computing Predictive Resource Management Framework Using Hidden Markov Model. In *2022 5th Conference on Cloud and Internet of Things (CIoT)*. 205–212. https://doi.org/10.1109/CIoT53061.2022.9766809

[2] Celia A. Baumhoer, Andreas J. Dietz, C. Kneisel, and C. Kuenzer. 2019. Automated Extraction of Antarctic Glacier and Ice Shelf Fronts from Sentinel-1 Imagery Using Deep Learning. *Remote Sensing* 11, 21 (2019). https://doi.org/10.3390/rs11212529

[3] Jonas Eberle, Maximilian Schwinger, and Julian Zeidler. 2023. Challenges in the development of the EO Exploitation Platform terrabyte. In *Proceedings of the 2023 Conference on Big Data from Space (BiDS'23) – From foresight to impact* (Vienna, Austria). Publications Office of the European Union, 97–100. https://doi.org/10.2760/46796

[4] German Aerospace Center (DLR). [n. d.]. terrabyte. https://www.dlr.de/eoc/terrabyte. Accessed: 2023-12-04.

[5] Piyush Gupta, Shashidhar G Koolagudi, Rahul Khanna, Mrittika Ganguli, and Ananth Narayan Sankaranarayanan. 2015. Analytic technique for optimal workload scheduling in data-center using phase detection. In *5th International Conference on Energy Aware Computing Systems & Applications*. IEEE, 1–4.

[6] Arijit Khan, Xifeng Yan, Shu Tao, and Nikos Anerousis. 2012. Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium*. 1287–1294. https://doi.org/10.1109/NOMS.2012.6212065

[7] Igor Klein, Ursula Gessner, Andreas J. Dietz, and Claudia Kuenzer. 2017. Global WaterPack – A 250m resolution dataset revealing the daily dynamics of global inland water bodies. *Remote Sensing of Environment* 198 (2017), 345–362. https://doi.org/10.1016/j.rse.2017.06.045

[8] David Buchaca Prats, Josep Lluís Berral, and David Carrera. 2017. Automatic generation of workload profiles using unsupervised learning pipelines. *IEEE Transactions on Network and Service Management* 15, 1 (2017), 142–155.

[9] Ivo Rohwer. 2024. Resource Demand Profiling of Monolithic Workflows. https://www.codeocean.com/. https://doi.org/10.24433/CO.0301206.v1

[10] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. 2011. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In *2011 IEEE 4th International Conference on Cloud Computing*. 500–507. https://doi.org/10.1109/CLOUD.2011.42

[11] Frank Thonfeld, Ursula Gessner, Stefanie Holzwarth, Jennifer Kriese, Emmanuel Canova, Juliane Huth, and Claudia Kuenzer. 2022. A First Assessment of Canopy Cover Loss in Germany's Forests after the 2018–2020 Drought Years. *Remote Sensing* 14 (01 2022), 562. https://doi.org/10.3390/rs14030562

[12] Anna Wendleder, Daniel Abele, Martin Huber, Birgit Wessel, Dennis Kaiser, John Truckenbrodt, Peter Friedl, Sandro Groth, Florian Fichtner, and Jonas Eberle. 2023. Sentinel-1 Normalized Radar Backscatter processing on the High-Performance Data Platform terrabyte. In *IGARSS 2023*. https://elib.dlr.de/196780/

[13] Da Yu Xu, Shan Lin Yang, and Ren Ping Liu. 2013. A mixture of HMM, GA, and Elman network for load prediction in cloud-oriented data centers. *Journal of Zhejiang University: Science C* 14, 11 (Nov. 2013), 845–858. https://doi.org/10.1631/jzus.C1300109

[14] Qi Zhang, Mohamed Faten Zhani, Shuo Zhang, Quanyan Zhu, Raouf Boutaba, and Joseph L. Hellerstein. 2012. Dynamic Energy-Aware Capacity Provisioning for Cloud Computing Environments. In *Proceedings of the 9th International Conference on Autonomic Computing* (San Jose, California, USA) *(ICAC '12)*. Association for Computing Machinery, New York, NY, USA, 145–154. https://doi.org/10.1145/2371536.2371562

# 12th International Workshop on Load Testing and Benchmarking of Software Systems: LTB'24 Chairs' Welcome

Marios Fokaefs
fokaefs@yorku.ca
York University
Toronto, Canada

Filipe Oliveira
filipe@redis.com
Redis
Lisbon, Portugal

Naser Ezzati-Jivan
nezzati@brocku.ca
Brock University
St.Catharines, Canada

LTB 2024
Load Testing & Benchmarking

## ABSTRACT

It is our great pleasure to welcome you to the twelfth edition of the International Workshop on Load Testing and Benchmarking of Software Systems – LTB 2024, (https://ltb2024.github.io/). This one-day workshop brings together software testing and software performance researchers, practitioners, and tool developers to discuss the challenges and opportunities of conducting research on load testing and benchmarking software systems, including theory, applications, and experiences. LTB 2024 includes 2 keynote talks, 4 research papers, and 2 industry presentations. The topics cover performance of serverless computing, performance and load testing, performance-driven culture, workload generation, workload tracing, benchmarking, and performance verification.

We warmly welcome attendees to attend our keynote, industry, and research talks:

- [Keynote] *Improving Software Quality Using AIOps.* Wahab Hamou-Lhadj (Professor of the Department of Electrical and Computer Engineering at Concordia University).
- [Keynote] *Scaling Performance Testing to Millions of Distinct Results.* David Daly (Staff Engineer – Developer Productivity at MongoDB).
- [Industry] *How to Sell Performance Test Results To a Diverse Crowd.* René Schwietzke (Xceptance).
- [Industry] *Performance Testing Transformation.* Alexander Podelko (Amazon).
- [Research] *Fastcrypto: Pioneering Cryptography Via Continuous Benchmarking.* Konstantinos Chalkias, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Joy Wang, and Alberto Sonnino.
- [Research] *Overhead Comparison of Instrumentation Frameworks.* David Georg Reichelt, Lubomir Bulej, Reiner Jung, and André van Hoorn.
- [Research] *Exemplary Determination of Cgroups-Based QoS Isolation for a Database Workload.* Simon Volpert, Sascha Winkelhofer, Stefan Wesner, Daniel Seybold, and Jörg Domaschka.
- [Research] *Self-Service Performance Testing Platform for Autonomous Development Teams.* Oleksandr Kachur and Aleksei Vasilevskii.

Putting together LTB'24 was a team effort. We first thank the authors for providing the content of the program. We are grateful to the program committee, who worked very hard in reviewing papers and providing feedback to authors. Finally, we thank ICPE for hosting our workshop.

We hope that you will find this program interesting and thought-provoking and that the workshop will provide you with a valuable opportunity to share ideas with other researchers and practitioners from institutions around the world.

**ACM Reference Format:**
Marios Fokaefs, Filipe Oliveira, and Naser Ezzati-Jivan. 2024. 12th International Workshop on Load Testing and Benchmarking of Software Systems: LTB'24 Chairs' Welcome. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3629527.3651438

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651437

# Fastcrypto: Pioneering Cryptography Via Continuous Benchmarking

Kostas Kryptos Chalkias
Mysten Labs
San Francisco, USA
kostas@mystenlabs.com

Jonas Lindstrøm
Mysten Labs
Aarhus, Denmark
jonas@mystenlabs.com

Deepak Maram
Mysten Labs
New York, USA
deepak@mystenlabs.com

Ben Riva
Mysten Labs
Tel Aviv, Israel
benriva@mystenlabs.com

Arnab Roy
Mysten Labs
San Francisco, USA
arnab@mystenlabs.com

Alberto Sonnino
Mysten Labs
London, UK
University College London
London, UK
alberto@mystenlabs.com

Joy Wang
Mysten Labs
New York, USA
joy@mystenlabs.com

## ABSTRACT

In the rapidly evolving fields of encryption and blockchain technologies, the efficiency and security of cryptographic schemes significantly impact performance. This paper introduces a comprehensive framework for continuous benchmarking in one of the most popular cryptography Rust libraries, `fastcrypto`. What makes our analysis unique is the realization that automated benchmarking is not just a performance monitor and optimization tool, but it can be used for cryptanalysis and innovation discovery as well. Surprisingly, benchmarks can uncover spectacular security flaws and inconsistencies in various cryptographic implementations and standards, while at the same time they can identify unique opportunities for innovation not previously known to science, such as providing a) hints for novel algorithms, b) indications for mix-and-match library functions that result in world record speeds, and c) evidences of biased or untested real world algorithm comparisons in the literature.

Our approach transcends traditional benchmarking methods by identifying inconsistencies in multi-threaded code, which previously resulted in unfair comparisons. We demonstrate the effectiveness of our methodology in identifying the fastest algorithms for specific cryptographic operations like signing, while revealing hidden performance characteristics and security flaws. The process of continuous benchmarking allowed `fastcrypto` to break many crypto-operations speed records in the Rust language ecosystem.

A notable discovery in our research is the identification of vulnerabilities and unfair speed claims due to missing padding checks in high-performance Base64 encoding libraries. We also uncover insights into algorithmic implementations such as multi-scalar elliptic curve multiplications, which exhibit different performance gains when applied in different schemes and libraries. This was not evident in conventional benchmarking practices. Further, our analysis highlights bottlenecks in cryptographic algorithms where pre-computed tables can be strategically applied, accounting for L1 and L2 CPU cache limitations.

Our benchmarking framework also reveals that certain algorithmic implementations incur additional overheads due to serialization processes, necessitating a refined 'apples to apples' comparison approach. We identified unique performance patterns in some schemes, where efficiency scales with input size, aiding blockchain technologies in optimal parameter selection and data compression.

Crucially, continuous benchmarking serves as a tool for ongoing audit and security assurance. Variations in performance can signal potential security issues during upgrades, such as cleptography, hardware manipulation or supply chain attacks. This was evidenced by critical private key leakage vulnerabilities we found in one of the most popular EdDSA Rust libraries. By providing a dynamic and thorough benchmarking approach, our framework empowers stakeholders to make informed decisions, enhance security measures, and optimize cryptographic operations in an ever-changing digital landscape.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652266

## CCS CONCEPTS

• **Security and privacy** → **Cryptography**; • **Software and its engineering** → Software libraries and repositories; **Software development process management**.

**ACM Reference Format:**
Kostas Kryptos Chalkias, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Alberto Sonnino, and Joy Wang. 2024. Fastcrypto: Pioneering Cryptography Via Continuous Benchmarking. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3652266

## 1 INTRODUCTION

Cryptography plays a pivotal role in safeguarding the integrity and confidentiality of secure communication channels, decentralized applications, digital identity and authentication systems, among the others. In the last 10-15 years, the demand for secure and scalable blockchain solutions caused an exponentially increased need for comprehensive performance evaluations of the underlying cryptographic components, such as digital signatures, zero knowledge proofs, Merkle trees, regular or exotic encryption mechanisms, multi-party computations and randomness beacons. It is believed that blockchain research has advanced the cryptography space rapidly [20], offering some of the most robust and fastest implementations that are now reused outside web3 as well.

`Fastcrypto` [24] is one of the most recent and modern Rust [23] libraries focusing on high performance implementations of cryptographic primitives, typically required by blockchain applications. Although originally designed to provide all cryptographic functionality for the Sui[1] blockchain, it has been widely adopted by the cryptographic community, and is currently used in at least 167 other projects[2].

A few prominent examples of `fastcrypto`'s community usage include the following: (1) DB3 Network[3], which is a lightweight, permanent JSON document database for Web3. It is designed to store and retrieve data for decentralized applications built on blockchain technology, (2) Rooch Network[4], which is a fast, modular, secured, developer-friendly infrastructure solution for building Web3 Native applications, and (3) Fleek Network[5], which facilitates the deployment and running of performant, geo-aware decentralized web and edge services. These codebases typically use the base64, hashing, and signature algorithms from `fastcrypto`.

In order to meet the performance demands of a scalable blockchain that must process thousands of transactions per second, `fastcrypto` has been continuously and rigorously benchmarked through the entire development process. This has informed decision-making, in particular in the early stages of the development where many crucial and largely irreversible choices had to be made.

This paper gives examples of some actionable insights acquired through our benchmarking efforts while developing the `fastcrypto` library. These insights have been leveraged for both the refinement of the library itself, and the optimization of cryptographic operations within Rust and Move [11] language based blockchains. In some cases this also led to changes in external libraries. It is our hope that these insights may be useful for researchers or developers working on performance critical systems.

[1]https://sui.io/
[2]https://github.com/MystenLabs/fastcrypto/network/dependents
[3]https://db3.network/
[4]https://rooch.network/
[5]https://fleek.network

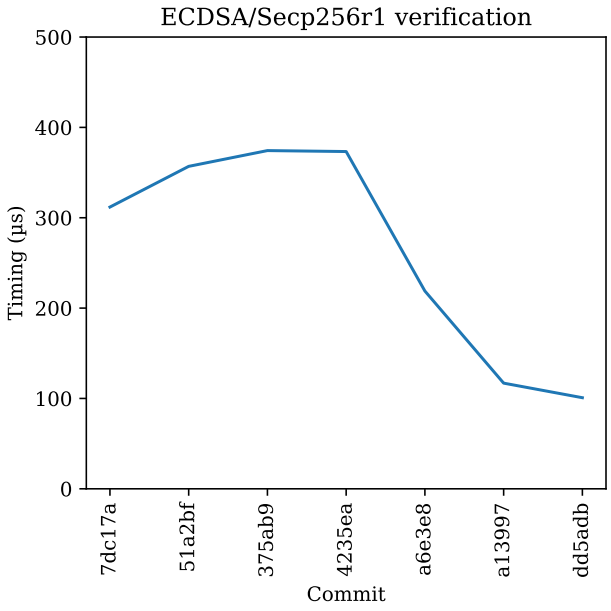



**Figure 1: Historic performance of a digital signature verification using the ECDSA signature scheme over the secp256r1 curve.**

## 2 METHOD

All cryptographic functions in the `fastcrypto` library are benchmarked continuously as part of the library's continuous integration workflow[6] and a report of the results are published online[7]. The report is generated using the criterion crate [3] and when applicable, functions are benchmarked with various input sizes and grouped together with similar functions to enable comparisons. Benchmarks are run sequentially and each measurement is run 100 times. The report contains the mean and standard deviation of the observed timings for further analysis. At the time of writing (January 2024), the report contains 109 different benchmarks.

The report contains historic data, allowing the detection of improvements or regressions in performance. As an example, Figure 1 shows a plot from the published report of the performance of verifying a digital signature using the ECDSA signature scheme over the secp256r1 (aka P-256) curve. This function has been improved several times which is reflected in the graph. These particular improvements are described in detail in section 3.1.3.

The data behind the report is published online in JSON format and may be analyzed using any statistical analysis tool. We have implemented a tool in Python[8] to utilize statistical libraries such as `numpy` [25] for more elaborate statistical analysis and plotting of the data. All plots in this paper were generated using this Python script.

[6]https://github.com/MystenLabs/fastcrypto/blob/main/.github/workflows/benchmarking.yml
[7]https://mystenlabs.github.io/fastcrypto/benchmarks/criterion/reports/
[8]https://github.com/jonas-lj/fastcrypto-analyzer

To identify bottlenecks when the cryptographic functions in `fastcrypto` are used elsewhere, we have made *Dummy* implementations of digital signatures and hash functions. These implementations use the same interfaces as the actual cryptographic functions and can be used in place of these. They are not cryptographically secure but are extremely fast, so when they are used in testing they allow a developer to identify where cryptographic operations are a bottleneck in their implementation.

## 3 CASE STUDIES

The continuous benchmarks have greatly influenced the decision-making in the development of the `fastcrypto` library and in how it is used in the Sui blockchain and later in other projects. In this section, we outline some of the insights we achieved through the benchmarks and their consequences for the development.

### 3.1 Picking the right dependencies and specs

*3.1.1 Signature aggregation can be catalytic.* The BLS signature scheme [13] allows multiple signatures generated under different public keys for the same message to be aggregated into a single signature which is valid only if all the individual signatures are valid [12]. In a blockchain setting, this has the potential to speed up validators' signature verification significantly, as it is possible to aggregate signatures and batch the verification, instead of individually submitting and verifying many independent signature payloads.

Signature schemes such as EdDSA and ECDSA are much faster than BLS for individual signatures (see Figure 2), but do not provide the same performance gain when signatures are batched, so choosing the right signature scheme requires careful assessment of performance [21].

Our benchmarks (see Figure 3) show that there are a number of signatures where verifying an aggregated BLS signature is the fastest option compared to EdDSA, and that using the fastest available implementations of EdDSA [26] and BLS [30], the break-even point is around 40-45 signatures.

Since BLS is used in the Sui blockchain to aggregate validators' signatures, this implies that if there are more than 45 validators, using BLS will be faster than EdDSA. At the time of writing, there are 106 validators in Sui, meaning that verifying aggregated BLS is about 2× faster than EdDSA, when all validators sign.

*3.1.2 Hash functions - in the mercy of hardware specs.* In blockchains, cryptographic hash functions are arguably the most used cryptographic primitive, so even though they are relatively fast functions they may eventually become a bottleneck.

The performance of all cryptographic hash functions are approximately linear in the input size for sufficiently large inputs, but there are subtle differences in performance because the data is processed in blocks of varying sizes and this difference is more noticeable for small inputs. Sui originally used the Sha3-256 hash function that Meta's Libra [1] project originally utilized, but after benchmarking alternatives it switched to Blake2b [6] which is almost 3× faster and more zero knowledge proof friendly.

A plot of the benchmarks is shown in Figure 4. Note that Sha256 is the fastest hash function here, but this is not the case on all

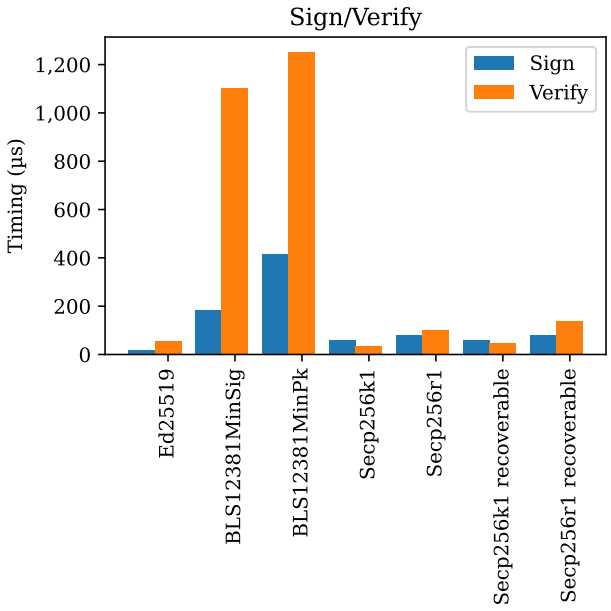



**Figure 2: Performance of signing and verifying a message using various digital signature schemes. Secp256k1 and Secp256r1 are variants of ECDSA.**

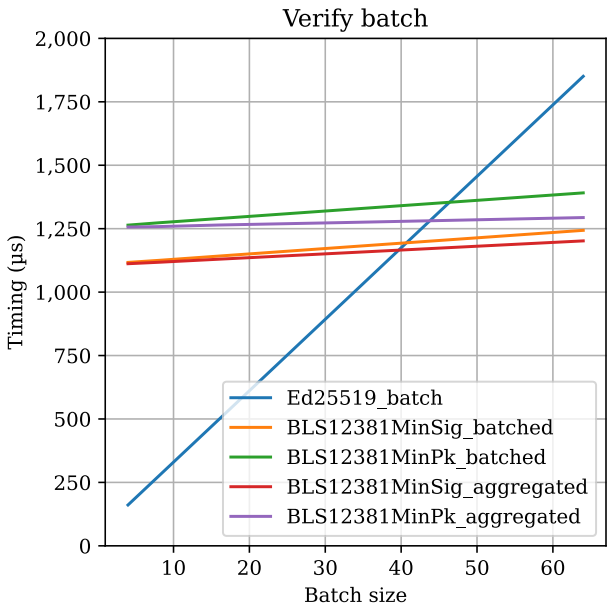



**Figure 3: Performance of batched verification of digital signatures using the EdDSA and BLS signature schemes.**

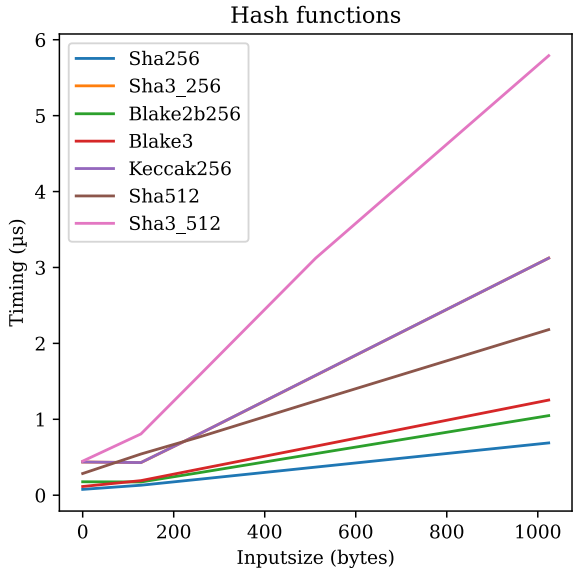



**Figure 4: Performance of cryptographic hash functions.**

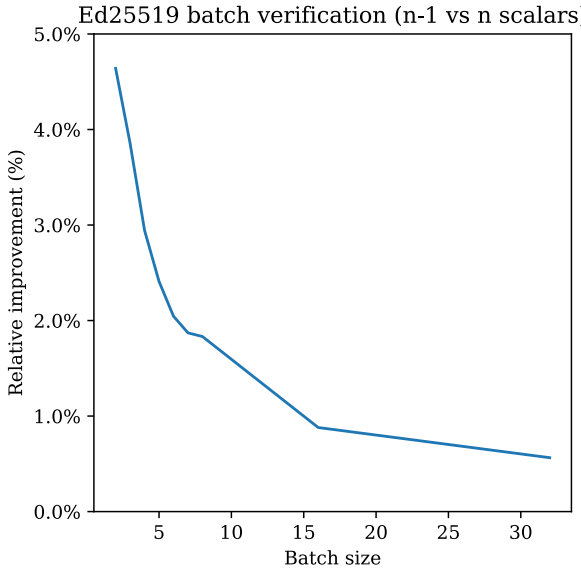



**Figure 5: Relative performance improvement between using $n$ and $n-1$ random scalars to verify a batch of $n$ Ed25519 signatures.**

platforms. This is evident, for example, from the benchmarks published by the Blake2 team[9] which shows that Sha256 is more than 2× slower than Blake2b, but we have also observed this in our benchmarks where the performance of Sha256 suddenly improved significantly between two runs without any changes in the software. We identified that this spike is due to recent updates in hardware for the cloud runner, because some hardware vendors have specialized CPUs to support Sha256 instructions; but running purely in software, Blake2b is faster.

We want to investigate this further and make our benchmarks fairer and more consistent, but it emphasizes the importance of benchmarking on a system similar to the production system because subtle differences (like CPU brand and model) can affect the performance significantly.

*3.1.3 Deserialization can be expensive in cryptography.* Many modern blockchains enable cryptographic agility for account signature key types. For instance in Sui blockchain, users may choose between a variety of signature schemes to sign their transaction[10]. This allows them to pick their favorite hardware wallet or their smartphone and store their keys securely. The default choice for the Sui blockchain (and many others) is EdDSA [10] over the ed25519 curve which was chosen based on high performance, determinism, adoption and standardization.

There are a few implementations of EdDSA in Rust, and comparing two popular crates (libraries) `ed25519-dalek` [15] and `ed25519-consensus` [26], which are backed by the same crypto arithmetic library, revealed some unexpected results, namely that the prior was much faster. Studying the source code closely showed that the difference was almost exclusively due to the fact that public keys in the latter are given in *compressed* serialized form, which is a representation where only one affine coordinate from the elliptic-curve point is given, meaning that the other coordinate has to be reconstructed before it can be used using a modular square root computation. This decompression operation is not for free. Accounting for this extra computation, the difference between the two libraries were then negligible. Some important lessons from this exercise are that a) we should be careful when comparing similar functions, (de)serialization can be expensive in cryptography, and b) there is a reason why some cryptographic libraries prefer one or the other, for instance the authors of `ed25519-consensus` explained that their method is safer when receiving public keys from the network, because the user does not need to take care of invalid keys before invoking the `signature.verify()` function; this is indeed a valid argument when keys are not cached or are unknown (typically the case in blockchain transactions).

*3.1.4 Asymptotic complexity does not always tell the truth.* EdDSA has a batched verification mode, where multiple signatures may be verified in a batch, giving some speed-up if enough signatures are verified together (see also section 3.1.1). While benchmarking this, we found an untapped potential optimisation: Typically, batch verification of $n$ EdDSA signatures requires sampling $n$ random scalars, but we found that $n-1$ is actually sufficient. For a small number of signatures, this gives a small speed-up as shown in Figure 5; this might make sense when we verify sponsored or atomic-swap transactions, where two accounts sign over the same transaction bytes.

## 3.2 Mix and Match Optimizations

*3.2.1 Optimize ECDSA over the P-256 curve.* As discussed in section 3.1.3 above, clients are allowed to choose among many signature

[9] https://www.blake2.net/
[10] https://docs.sui.io/guides/developer/sui-101/sign-and-send-txn

schemes when signing their transactions, but it turns out that some schemes are slower than others so to avoid that verifying signatures of a particular scheme becomes a bottleneck for the entire system, the signature schemes are benchmarked continuously. The information from the benchmarks may be used to encourage users to use the faster schemes, for example by using these schemes as default choice in wallet implementations, but also to identify where optimising an implementation will have the largest effect.

As an example, the ECDSA signature scheme [2] may be realised over different elliptic curves. Two commonly used curves are secp256k1 which is used by the Bitcoin[11] and Ethereum blockchains [31] and the secp256r1 or P-256 curve which was specified by NIST and is used, for example, by the secure hardware on iPhone[12]. Both of these are supported by the Sui blockchain and may be used by clients to sign their transactions.

Besides the choice of curve, there is no difference in the protocols for ECDSA over the two curves, but our benchmarks revealed that the fastest implementation of ECDSA over secp256k1 [27] is significantly faster than the fastest implementation of ECDSA over P-256 [28]. This motivated us to develop a new implementation of ECDSA over P-256 which uses a combination of faster elliptic curve arithmetic from Arkworks library [4] with a new, fast multi-scalar multiplication algorithm which requires some pre-computation. The optimised implementation verification for ECDSA over the P-256 curve is 5.5× faster, and is currently the fastest Rust implementation of ECDSA over the P-256 curve available.

Choosing the right number of pre-computed points for the multi-scalar multiplication required careful benchmarking, see figure 6. More pre-computed points (at least up to a certain limit) gives better performance but takes time and space. For our implementation, we use 256 points (each taking up 64 bytes) as default which gives a 68% improvement compared to not using multi-scalar multiplication at all and a 17.5% improvement compared to pre-computing only 16 points. Increasing the precomputation further to, say, 512 points would only give an 1.3% performance improvement, and for 1024 points, performance regresses, so 256 points was chosen as a compromise for our implementation. See Figure 6 for a plot of performance over number of pre-computed points.

*3.2.2 A faster Poseidon hash function.* The Poseidon hash function [16] is a hash function which is commonly used in zero-knowledge applications because it is easy to compute inside a zero-knowledge circuit. The Poseidon hash function is defined over a specific curve construction, and you need to use the same construction as for the zero-knowledge proof it is used in to get the performance benefit.

There are a few Rust implementations of the Poseidon hash, but not all implementations support all curve constructions. For our purpose, we needed to use the BN254 curve construction for zkLogin[13] [7] and only the poseidon-ark [5] crate supported this construction.

Benchmarking the zkLogin flow end-to-end revealed that computing the Poseidon hash took about 40% of the time so we decided

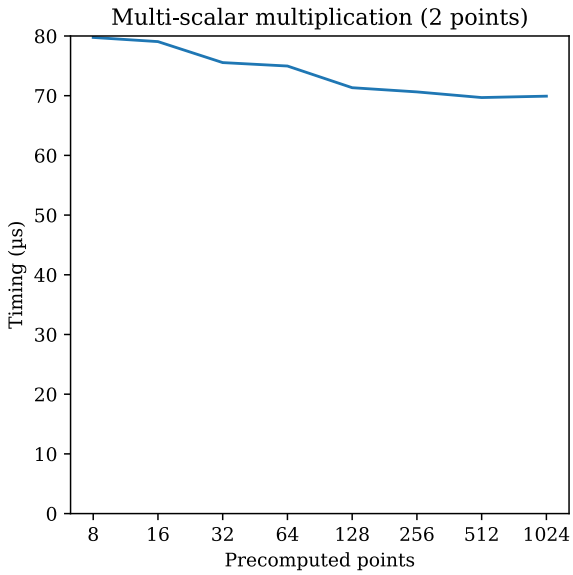



**Figure 6: Performance of windowed multi-scalar multiplication with two points on Secp256r1 where one is known in advance over the number of precomputed points. As a reference, a naive computation without any precomputation takes 175 µs.**

to see if we could optimise it. We found that there are faster implementations of the Poseidon hash function in Rust in particular the neptune [22] crate, but at the time the neptune crate only supported the BLS12-377 curve construction and not the BN254 construction we needed in zkLogin. Using neptune over BN254 required a few changes to the implementation which we contributed by submitting code to the official repository [14] before we could use it. The resulting implementation is almost 70% faster cutting of 25% of the total end-to-end flow for zkLogin (Figure 7).

*3.2.3 Combining dependencies for optimal performance.* `Fastcrypto` supports non-interactive zero-knowledge proofs using the Groth16 zk-SNARK construction [18] over two popular curves, namely the BN254 and BLS12-381 [8] curve constructions. Arkworks [4] have implementations of Groth16 for both of these constructions, but for the BLS12-381 construction the blst crate [30] provides a much faster implementation of the curve arithmetic, but does not provide any implementation of Groth16.

In `fastcrypto` we have combined Arkworks' implementation of Groth16 with the elliptic curve arithmetic from the blst crate to create a Groth16 implementation over BLS12-381 that is almost 2× faster than Arkworks implementation. To make this implementation efficient it was important to benchmark all steps of the algorithm independently, in particular the data conversions necessary to combine the blst and Arkworks libraries, to ensure that these conversions did not introduce a significant overhead. A performance

[11] https://en.bitcoin.it/wiki/Secp256k1
[12] https://developer.apple.com/documentation/cryptokit/p256/signing/ecdsasignature
[13] https://sui.io/zklogin
[14] https://github.com/lurk-lab/neptune/pull/236

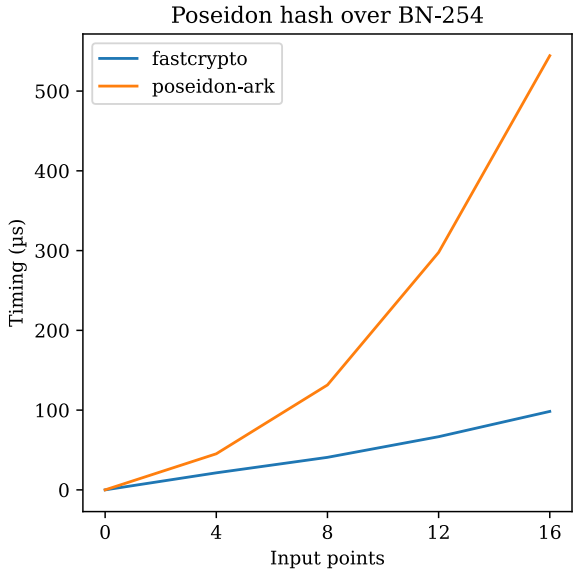



**Figure 7: Performance of computing the Poseidon hash over the BN254 curve construction for 0-16 input points using the `fastcrypto` implementation compared with the arkworks-rs crate.**

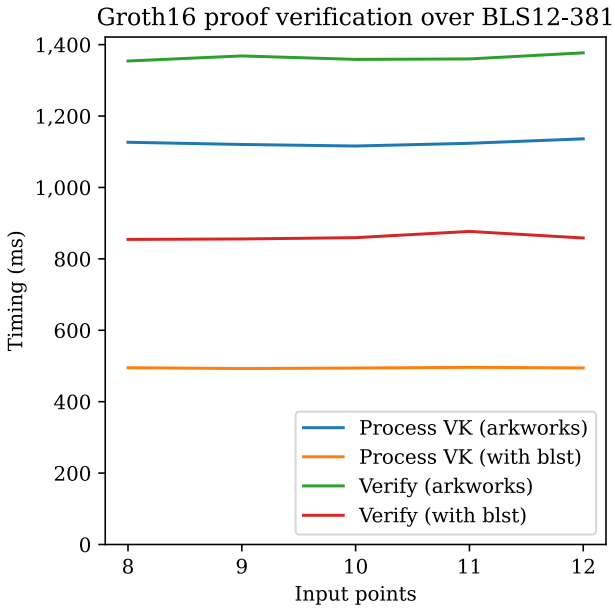



**Figure 8: Performance our implementation of Groth16 zk-proof verification vs. Arkworks' implementation. The performance is independent of the input size, as the plot also shows.**

comparison of our implementation with Arkworks' implementation is shown in Figure 8. Note that a full verification of a Groth16 zk-proof consists of processing the verification key *and* verifying the proof, but the processing of the verification key only have to happen once per circuit.

## 3.3 Errors and inconsistencies in dependencies

*3.3.1 Bug in base64 implementations.* `Fastcrypto` contains functions to encode data to and from base64 which is a very commonly used method to map binary data to ASCII characters, for example for use for serialization purposes. Implementing this, we tested out a few potential Rust crates to wrap in `fastcrypto` and benchmarked them on different input sizes.

The benchmarks revealed unexpectedly significant differences in performance between different libraries, and a closer study found that the difference was caused by some of the libraries not handling padding correctly. This inconsistency causes some libraries for base64 encoding to be incompatible, which is very unfortunate since base64 is often used for serialization and thus depends on portability. It also allows an attack vector on some systems because an attacker may utilize that different base64 strings are decoded into the same data to leverage an attack. This finding and a thorough description of the potential consequences has been published [14].

*3.3.2 Exploitable vulnerability in EdDSA libraries.* As previously mentioned, `fastcrypto` compares many implementations of the same signature schemes and then wraps the fastest or uses mix and match or applies extra expert optimizations. We realized that some exposed public functions for EdDSA *signing* were significantly slower than other implementations even when the libraries where backed by the same back-end arithmetic dependency. A closer look resulted in identifying one of the most spectacular exploitable cryptography vulnerabilities, not only in Rust, but as a domino effect in dozens of cryptographic libraries, a potential vulnerability that was featured in the news [9] and for which a RUSTSEC fix was issued [29]. In short, many libraries, including the popular `ed25519-dalek` expose a `sign` function that additionally takes the public key as an input, and not only the private key and the message, which is the typical architecture in digital signature APIs. The reason behind this implementation design is speculated to be related to performance optimizations, because that addition allowed the function to avoid computing the public key (from the private) internally, and hence it was faster due to avoiding deserialization and other operations we highlighted in section 3.1.3. Note that exploiting such a function could result in private key leakage, an attack that we published as "Double Public Key Signing Function Oracle Attack on EdDSA Software Implementations" [17].

*3.3.3 Unwanted parallelization for BLS verification.* As with the base64 bug described above, surprising benchmark results are often a hint that some libraries are behaving unexpectedly. In an earlier version of `fastcrypto`, both BLS signature verification [13] over the BLS12-381 and the BLS12-377 constructions [8] were supported. However, BLS12-377, which used the Arkworks [4] implementation, was significantly slower than BLS12-381 which uses the `blst` [30] crate. Analysing this further, we noticed that blst by default allows multi-threaded computations. However, when allowing BLS12-377 to do the same, we got a *regression* in performance. It is unclear why this was the case, but the benefit of using multiple threads for

BLS signatures is small (around 25% for `blst`), so if the threads are not managed tightly the small potential improvement from using multiple threads will be lost and performance will regress instead.

In our case, where the primary usage is to verify transaction signatures on the Sui blockchain, we decided to only allow single-threaded verification, because Sui is already a multi-threaded application, and allowing multiple threads for signature verification alone will complicate the thread management for Sui.

*3.3.4 Unwanted parallelization for Groth16 proving.* An important dependency for zkLogin is rapidsnark [19], a software library that leverages assembly code to speed up the process of generating a Groth16 zero-knowledge proof. It is well known that proving is one of the main remaining bottlenecks for zero-knowledge proofs. In order to optimize as much as possible, rapidsnark provides a server-like interface to process several requests at once. However, our testing revealed that the results returned by the prover under simultaneous requests was often erroneous. This was likely due to improper handling of state between threads resulting in one of the threads over-writing results of another.

Further inspection revealed that rapidsnark already utilizes available parallelism to generate a single zero-knowledge proof. Given this scenario, we decided to modify the library to disable the multi-request feature. We adopt a simpler strategy to handle simultaneous requests: scale the deployment horizontally by adding multiple machines.

We leave it for future work to conduct thorough benchmarks to identify if processing simultaneous requests on a single machine is actually useful. We suspect that it may only be useful on machines with a lot of parallelism or cores. Also note that when the number of cores is not high, then there is a risk of performance regression, that is, processing a request takes more time if there are simultaneous requests than otherwise, which is undesirable in most user-facing applications.

## 3.4 Continuous benchmarks

The life cycle of our primitives, from initial prototyping to production readiness, extends over several months. The initial implementation is typically unoptimized, emphasizing simplicity and accompanied by basic unit tests. Subsequent cycles focus on refining the primitive until it reaches a state suitable for performance measurements. Various evaluations are integral to this process:

- Local Benchmarks. These involve extensive testing with a diverse range of inputs. These benchmarks serve dual purposes—facilitating rapid development and ensuring progress across optimization cycles.
- Continuous Integration (CI) Tests. These tests are vital for ensuring that any future changes do not introduce performance regressions. They act as a safeguard against unintended setbacks in the optimization journey. This step is crucial as recent changes in sub-components of the library can impact the performance of primitives implemented and benchmarked in the past.

Continuous tests also guarantee accurate and up-to-date benchmark outcomes. They ensure that the latest performance measurements are reported, even in primitives implemented and benchmarked long ago.

# 4 CONCLUSION AND FUTURE WORK

In the development of the `fastcrypto` library, continuous benchmarking has been a crucial tool in identifying bottlenecks and in qualifying the decision-making, notably when choosing what protocols and software libraries to use, but the benchmarks have in some cases also revealed unexpected insights into the inner workings of dependencies and even revealed critical bugs.

The benchmarks are published online and may also be used by developers to compare implementations or to compare with their own implementations. We have published a Python script to analyse the published data [15], and we hope to integrate this script with our continuous integration workflow, e.g. to detect performance regressions automatically. The measurements show a large variation, probably because they are run on a cloud service, and we would also like to explore how to make measurements more consistent.

All in all, continuous benchmarks are more than a performance metric tool, it can be an excellent tool to identify vulnerabilities and allow for novel protocol designs and even world record implementations.

# REFERENCES


[1] Zachary Amsden, Ramnik Arora, Shehar Bano, Mathieu Baudet, Sam Blackshear, Abhay Bothra, G Cabrera, C Catalini, K Chalkias, E Cheng, et al. 2019. The libra blockchain. *URl: https://developers. libra. org/docs/assets/papers/the-libra-blockchain. pdf* (2019).

[2] X9 ANSI. 1999. 62: public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ecdsa). *Am. Nat'l Standards Inst* (1999).

[3] Jorge Aparicio and Brook Heisler. 2024. criterion.rs: Statistics-driven micro-benchmarking library. https://github.com/japaric/criterion.rs.

[4] Arkworks. 2024. arkworks-rs. https://github.com/arkworks-rs/.

[5] arnaucube. 2024. poseidon-ark. https://github.com/arnaucube/poseidon-ark.

[6] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2013. BLAKE2: simpler, smaller, fast as MD5. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security* (Banff, AB, Canada) *(ACNS'13)*. Springer-Verlag, Berlin, Heidelberg, 119–135. https://doi.org/10.1007/978-3-642-38980-1_8

[7] Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, and Joy Wang. 2024. zkLogin: Privacy-Preserving Blockchain Authentication with Existing Credentials. arXiv:2401.11735 [cs.CR]

[8] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. 2003. Constructing Elliptic Curves with Prescribed Embedding Degrees. In *Security in Communication Networks*, Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 257–267.

[9] Ben Dickson. 2022. Dozens of cryptography libraries vulnerable to private key theft. The Daily Swig: https://portswigger.net/daily-swig/dozens-of-cryptography-libraries-vulnerable-to-private-key-theft.

[10] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2, 2 (2012), 77–89. https://doi.org/10.1007/s13389-012-0027-1

[11] Sam Blackshear, Evan Cheng, David L Dill, Victor Gao, Ben Maurer, Todd Nowacki, Alistair Pott, Shaz Qadeer, Dario Russi Rain, Stephane Sezer, et al. 2019. Move: A language with programmable resources. *Libra Assoc* (2019), 1.

[12] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology — EUROCRYPT 2003*, Eli Biham (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 416–432.

[13] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *Advances in Cryptology — ASIACRYPT 2001*, Colin Boyd (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 514–532.

[14] Konstantinos Chalkias and Panagiotis Chatzigiannis. 2022. Base64 Malleability in Practice. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security* (Nagasaki, Japan) *(ASIA CCS '22)*. Association for Computing Machinery, New York, NY, USA, 1219–1221. https://doi.org/10.1145/3488932.3527284


[15] https://github.com/jonas-lj/fastcrypto-analyzer

[15] dalek cryptography. 2024. ed25519-dalek. https://github.com/dalek-cryptography/ed25519-dalek.
[16] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *USENIX Security Symposium*. https://api.semanticscholar.org/CorpusID:221069468
[17] Sam Grierson, Konstantinos Chalkias, and William J Buchanan. 2023. Double Public Key Signing Function Oracle Attack on EdDSA Software Implementations. *arXiv preprint arXiv:2308.15009* (2023).
[18] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. 305–326. https://doi.org/10.1007/978-3-662-49896-5_11
[19] iden3. 2024. rapidsnark. https://github.com/iden3/rapidsnark.
[20] Kostas Kryptos. 2023. Blockchain research has advanced systems and cryptography. https://twitter.com/kostascrypto/status/1626983601572302848.
[21] Zhuolun Li, Alberto Sonnino, and Philipp Jovanovic. 2023. Performance of EdDSA and BLS Signatures in Committee-Based Consensus. In *Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*.
[22] lurk-lab. 2024. neptune. https://github.com/lurk-lab/neptune.
[23] Nicholas D Matsakis and Felix S Klock. 2014. The rust language. *ACM SIGAda Ada Letters* 34, 3 (2014), 103–104.
[24] Mysten Labs. 2024. fastcrypto. https://github.com/MystenLabs/fastcrypto.
[25] NumPy Team. 2024. Numpy. https://numpy.org.
[26] Penumbra. 2024. ed25519-consensus. https://github.com/penumbra-zone/ed25519-consensus.
[27] Rust Bitcoin Community. 2024. rust-secp256k1. https://github.com/rust-bitcoin/rust-secp256k1/.
[28] RustCrypto. 2024. p256. https://github.com/RustCrypto/elliptic-curves/tree/master/p256.
[29] Rustsec. 2022. Double Public Key Signing Function Oracle Attack on ed25519-dalek. RUSTSEC-2022-0093: https://rustsec.org/advisories/RUSTSEC-2022-0093.
[30] Supranational. 2024. blst. https://github.com/supranational/blst.
[31] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. , 32 pages.

# Exemplary Determination of Cgroups-Based QoS Isolation for a Database Workload

Simon Volpert
simon.volpert@uni-ulm.de
Ulm University
Institute of Information Resource Management
Ulm, Germany

Sascha Winkelhofer
sascha.winkelhofer@gini.net
Gini GmbH
Munich, Germany

Stefan Wesner
wesner@uni-koeln.de
University of Cologne
Cologne, Germany

Daniel Seybold
Jörg Domaschka
daniel.seybold@benchant.com
joerg.domaschka@benchant.com
BenchANT GmbH
Ulm, Germany

## ABSTRACT

An effective isolation among workloads within a shared and possibly contended compute environment is a crucial aspect for industry and academia alike to ensure optimal performance and resource utilization. Modern ecosystems offer a wide range of approaches and solutions to ensure isolation for a multitude of different compute resources. Past experiments have verified the effectiveness of this resource isolation with micro benchmarks. The effectiveness of Quality of Service (QoS) isolation for intricate workloads beyond micro benchmarks however, remains an open question.

This paper addresses this gap by introducing a specific example involving a database workload isolated using Cgroups from a disruptor contending for CPU resources. Despite the even distribution of CPU isolation limits among the workloads, our findings reveal a significant impact of the disruptor on the QoS of the database workload. To illustrate this, we present a methodology for quantifying this isolation, accompanied by an implementation incorporating essential instrumentation through Extended Berkeley Packet Filter (eBPF).

This not only highlights the practical challenges in achieving robust QoS isolation but also emphasizes the need for additional instrumentation and realistic scenarios to comprehensively evaluate and address these challenges.

## CCS CONCEPTS

• **General and reference** → **Performance**; • **Computing methodologies** → *Modeling methodologies.*

## KEYWORDS

Isolation, Performance, eBPF, Benchmarking, Cloud, DBMS

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652267

**ACM Reference Format:**
Simon Volpert, Sascha Winkelhofer, Stefan Wesner, Daniel Seybold, and Jörg Domaschka. 2024. Exemplary Determination of Cgroups-Based QoS Isolation for a Database Workload. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3652267

## 1 INTRODUCTION

In the ever-evolving landscape of computing, the paradigm shift toward cloud computing and larger-scaled compute environments has revolutionized the way organizations deploy, manage, and utilize computing resources. Cloud computing, in particular, offers unparalleled scalability, flexibility, and cost-effectiveness, enabling businesses and scientists to work on challenges that were unattainable without it [14, 17]. However, the shared nature of resources in such environments introduces inherent challenges, necessitating robust mechanisms to ensure isolation among disparate workloads and tenants. These challenges can be imposed by the desire to consolidate physical hardware, overbooking or overcommitting as a business model, or by misbehaving disruptive tenants acting as "noisy neighbors".

There is a wide range of solutions that aim to solve these isolation challenges. One aspect towards a solution is various virtualization technologies. These range from classic hypervisor-based implementations over manifold container-based solutions towards more recent developments in the concept of application sandboxing. Many of them pursue different strategies to achieve adequate isolation; however, they do share some commonalities. A frequently used strategy is the utilization of Cgroups [7, 23].

Cgroups are provided by the Linux kernel. They enable an operator to distribute processes into groups and subsequently assign resource limits to those groups. These mechanisms have proven to work very well, specifically when solely observing the isolated and limited resource. The patterns of resource usage of real-world applications are often more complex [5]. Their QoS is not necessarily directly dependent on a few distinct resources, as it is a measure of end-to-end performance that inherently involves any amount of resources [4, 16].

In this paper, we focus on Cgroup-based CPU isolation. For this specific case, we investigate whether one tenant's QoS is impacted by another disrupting tenant, even though their CPU limits are evenly shared with no overbooking in place. With this, we aim at answering the following two research questions:

**RQ 1 (Isolation measurement).** *How can QoS isolation be challenged in complex deterministic scenarios?*

**RQ 2 (Cgroup sufficiency).** *Is Cgroup-based isolation enough for reliant QoS isolation?*

Answering these questions, we provide several contributions. First, we provide a strategy to measure isolation between two tenants considering the impact on QoS. Second, we suggest metrics that quantify the degree of isolation. Finally, we provide a tool developed for this work that enables low-overhead instrumentation of compute resources for isolated process trees.

The remainder of this paper is structured as follows. In section 2 we discuss the fundamentals of this work. This includes eBPF profiling, Cgroups and a discussion of isolation and its quantification for QoS. This is followed by a description of the methodology applied in section 3 and lays the foundation for the answer to **RQ 1**. The methodology is followed by important details of the implementation in section 4. It gives a brief overview of the technologies involved in the experimental setup and the workflow of measured scenarios. The final results in section 5 discuss the observations and in this process answers **RQ 2**. We close with a review of related work in section 6 and a final summary in section 7.

## 2 BACKGROUND

This section describes important background aspects for the subsequent progression of this work. This includes low-overhead instrumentation, Cgroups, and isolation considerations.

### 2.1 Linux Profiling with eBPF

The Linux profiling subsystem efficiently gathers and collects performance data, enabling developers and operators to pinpoint and enhance resource utilization patterns. Retrieving these comes with a performance penalty depending on the method of accessing it.

eBPF facilitates the execution of verified code within a dedicated Virtual Machine (VM) integrated in the Linux kernel, extending the capabilities of the original Berkeley Packet Filter (BPF) [13]. Beyond executing functions upon receiving network packets, eBPF can observe and respond to various event sources as part of the Linux profiling subsystem, including Performance Monitoring Counters (PMCs), tracepoints, and both kernel and user functions.

Although these events are not technically part of eBPF, it provides an accessible means of leveraging them. Specifically, the instrumentation and processing of profiling data directly within the kernel space can reduce instrumentation overhead, since frequent interactions with kernel and userspace are kept to a minimum.

The typical lifecycle of an eBPF program is depicted in fig. 1, as presented by Gregg [6]. As depicted here, a typical first step is the *(i)* generation of BPF byte-code by arbitrary eBPF tooling. Upon this generation, the byte-code is *(ii)* loaded into the kernel for a verifying step before being passed to the eBPF VM. For exchanging data between Kernel- and userspace, the *(iii)* perf_output and *(iii)* async read channels can be utilized.

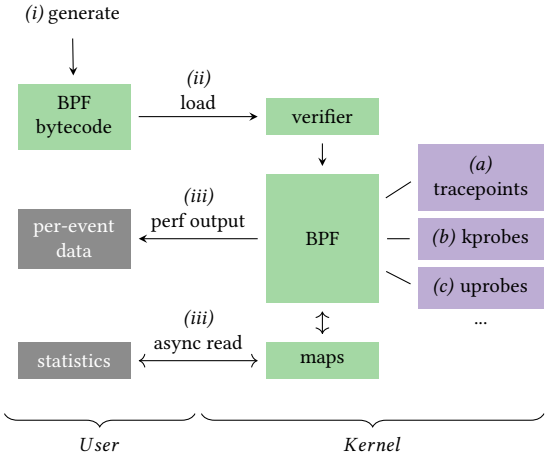



**Figure 1: eBPF internals and Linux instrumentation according to [6]**

Within the scope of this work, we are employing instrumentation on *(a)* Tracepoints. Tracepoints are static points of kernel instrumentation [19], established and implemented by kernel developers to trigger an event upon a specific call. They also incorporate hardware-specific counters, such as CPU cycles per core since boot time.

eBPF based instrumentation is naturally tightly coupled with the currently loaded Linux kernel. The BPF Type Format (BTF) aims to improve the portability of eBPF based tools by providing a metadata format, which encodes debug information related to the functions and structures of the kernel referenced in the eBPF programs. The profiling tool trac[1], developed during this work, utilizes this format. The tool itself is in an ongoing development phase.

This section only briefly outlines eBPF and Linux profiling, with a more detailed exposition available in the previous work of fellow authors [2, 20].

### 2.2 Cgroups

Control groups[2] are a Linux feature that enables precise control over the utilization of various system resources [8]. The Linux kernel ensures that the processes assigned to such a group adhere to the limits specified for the Cgroup. Additionally, Cgroups can be unique, shared, and nested, essentially creating a hierarchical structure.

Cgroups offer powerful measures to control, limit, and possibly isolate resources. Used in conjunction with namespaces, they act as an essential enabler for virtualization, particularly in the context of container virtualization [20].

The Cgroups project underwent a significant restructuring effort, resulting in the recent release of Cgroups v2. This effort was first merged into the kernel with version 4.5 and is able to fully replace v1

[1] https://github.com/omi-uulm/trac
[2] https://man7.org/linux/man-pages/man7/cgroups.7.html

since kernel version 5.6 [3]. At the time of writing, the list of Cgroup controllers include *(i)* cpu, *(ii)* cpuset, *(iii)* freezer, *(iv)* hugetlb, *(v)* io, *(vi)* memory, *(vii)* perf, *(viii)* pids, and *(ix)* rdma.

This work focuses on the usage of the *(i)* CPU controller implemented by Cgroups v2. It enables setting a limit on the number of CPU cycles per second.

## 2.3 Isolation Terminology

Isolation is a condition that occurs when two workloads share a resource and compete for it. The degree to which they interfere with each other characterizes isolation. If their influence on each other is distinctive, the isolation is considered low, and vice versa. This concept is discussed by several authors [10, 12, 22]. This study follows the definition of isolation provided by Krebs et al. who define:

Definition 1 (Isolation). *Performance isolation is the ability of a system to ensure that tenants working within their assigned quota (i.e., abiding tenants) will not suffer performance degradation due to other tenants exceeding their quotas (i.e., disruptive tenants).*

In a similar context, particularly in cloud computing, the term "noisy neighbor" is often used in related literature. This term refers to a disruptive tenant that adversely affects another tenant. According to the definition provided by Longbottom [11], a noisy neighbor is described as follows:

Definition 2 (Noisy Neighbor). *A workload within a shared environment is utilizing one or more resources in a way that it impacts other workloads operating around it.*

## 2.4 QoS Isolation Quantification

Performance degradation is a measure of how strong an abiding workload $W_a$ is affected by a disruptive workload $W_d$. It can be determined as "performance loss rate" $I_{plr}$ [9, 12, 18, 22].

$$I_{plr} = \frac{|W_{a_1} - W_{a_2}|}{W_{a_1}} \tag{1}$$

Here $W_{a_1}$ represents a workload in an undisrupted environment, whereas $W_{a_2}$ represents the same workload impacted by a disruptive workload $W_d$.

Krebs et al. extends this simple model with one specifically targeted at QoS isolation determination [10]. We apply and slightly adapt this model to fit our measured parameters.

Taking eq. (1) as a basis, we can determine the actual performance ratio $q_{W_a}$ and $q_{W_b}$ by calculating $1-I_{plr}$. This leads to the simplified eq. (2) and eq. (3).

$$q_{W_a} = \frac{W_a}{W_{a_{ref}}} \tag{2}$$

$$q_{W_d} = \frac{W_d}{W_{d_{ref}}} \tag{3}$$

Using eq. (2) and eq. (3) we can then determine the remaining relative performance $\rho$ at a certain $q_{W_d}$ as $q_{W_a}$.

$$\rho(q_{W_d}) = q_{W_a} \tag{4}$$

**Table 1: Scenarios**

| name | $W_a$ | $l_a$ | $W_d$ | $l_d$ |
|---|---|---|---|---|
| *(i)* baseline | 100 % | 50 % | 0 % | 50 % |
| *(ii)* harmony | 100 % | 50 % | 0-100 % | 50 % |

As Krebs et al. further states, these kind of values represent only a distinct point where the disruption is to a specific degree. To address this, we can try to reduce the resulting series of eq. (4) to a single isolation metric $I$.

An approach is to limit the number of samples $q_{W_d}$ to $m$ equidistant points and subsequently compute their arithmetic mean:

$$I_{avg} = \frac{\sum \rho(q_{W_d})}{m} \tag{5}$$

As this likely neglects the maximum amount of degradation, we can further derive another metric that describes the maximum isolation impact $I_{max}$ as follows:

$$I_{max} = \frac{\min(q_{W_a})}{\arg\min\limits_{q_{W_d}}(q_{W_a})} \tag{6}$$

Naturally, employing either eq. (5) or eq. (6) might overlook the inherent curve of $I_{QoS}$, potentially introducing a bias to the outcome. Further considerations on deriving a metric that avoids this are left for future work.

# 3 METHOD

This section presents the method behind the conducted experiments and thus elaborates on the scenarios, instrumentation, and isolation quantification. These aspects are adapted from previous work [20, 21].

*Goal.* As mentioned in section 1 we aim to measure the Cgroup QoS isolation for the CPU resource. According to section 2.4 we need at least two distinct measurements to analyze the isolation capability of a technology. One being the reference workload in an uncontended environment, and the other being the same workload under contention.

*Scenarios.* As we are interested in whether a QoS-based isolation is as high as a specific isolation for a certain resource, we choose an appropriate isolation scenario. Earlier work has shown that this is the case for fairly distributed resources where no overbooking, overcommitting, or aggressive resource stealing happens [20, 21]. Volpert et al. show that this is particularly true for the "harmony" scenario.

Therefore, this work analyzes the isolation of two scenarios: *(i)* baseline and *(ii)* harmony. These are itemized in table 1

Here, $W_a$ and $W_d$ describe the workload performed within their respective imposed limits $l_a$ and $l_d$. $W_a$ is considered static in both scenarios and is instrumented regarding its consumed resources and QoS status. It is further supposed to resemble a realistic workload and is thus realized as a macro or synthetic benchmark [9]. For the *(ii)* harmony scenario, $W_d$ gradually increases over time and is also instrumented for its consumed resources. Its purpose is to specifically stress the single resource that is being isolated.

*Instrumentation.* Again, the resource instrumentation approach follows the principles outlined in previous work by the authors [20, 21]. In summary, it is independent of isolation technology and performed outside of the isolation group. This is achieved with eBPF.

*Isolation quantification.* In section 2.4, we introduce and briefly examine metrics for quantifying QoS isolation. Utilizing eBPF and QoS metrics reported by $W_a$ we can quantify the isolation at specific degrees of contention by $W_d$.

# 4 EXPERIMENT DESIGN

In this section, we describe the abstract workflows of the experiments. These are followed by a presentation and reasoning behind the choices for the tools and instrumentation points selected.

## 4.1 Experiment workflow

As described in section 3, the experimental workflow follows two scenarios. The execution of a scenario is highlighted in fig. 2

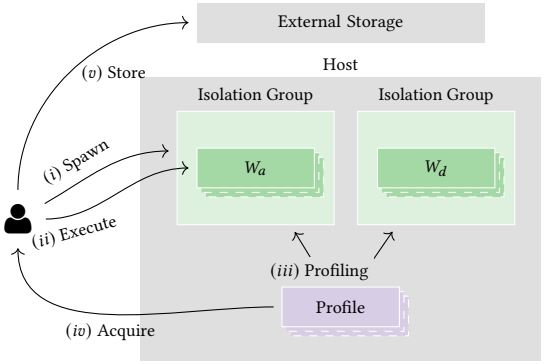



**Figure 2: Flow of an abstract measurement**

The process begins with *(i)* the initialization of an isolation group. In this phase, *(ii)* load is generated by $W_a$ and $W_b$. The *(iii)* profiling process on the host system is initiated concurrently. This profiling monitors the isolation groups. Upon completion, data is *(iv)* collected and *(v)* stored on external storage.

Each run takes 5 minutes and is repeated 3 times. After each run, the whole physical systems are reset and pruned to guarantee no unintended side effects by residue of past experiments and improving reproducibility.

## 4.2 Approach and Implementation

The following briefly iterates over the actual implementation of the method as described in section 3 is realized.

*Load generation.* As stated in section 3 we need two distinct workloads $W_a$ and $W_d$. $W_a$ is supposed to act as a realistic workload. Here, we choose to run a YCSB[3] benchmark on a remote host against a Postgres database [1]. The throughput in operations per second and thus the QoS workload $W_a$ is determined for this databse. For the sake of simplicity, we choose an insert-only workload stressing the database for 5 minutes. In that 5 minutes, YCSB tries to execute 100, 000, 000 inserts of 500 bytes with 90 threads. After its run-time, it reports a list of all operations with timestamp and latency. Operations per second can be derived by resampling to a desired frequency and counting the operations. These operations per second are considered to be the QoS metric of $W_a$.

The Postgres database is continually instrumented with respect to its CPU cycles and operations per second. Its configuration is generated by PGtune[4] optimizing Postgres with half of the total resources available on the physical server as described in section 5.1[15].

$W_d$ is considered to be a micro benchmark that continuously stresses the CPU. We use the `stress-ng` implementation to realize that load. It is set up such that it increases its utilization over time, until it fully utilizes its granted resources. To achieve a linear load generation behavior, we partition this load generation into multiple intervals with configurable resolution.

Assuming ideal isolation, the measures of both workloads resemble a progression, as illustrated in 3.

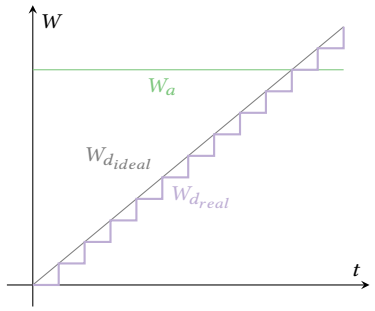



**Figure 3: Load generation**

*Instrumentation.* Since we focus on CPU isolation, we select an instrumentation point as outlined in section 2.1 that gives a detailed view on CPU utilization. Modern CPUs provide hardware-based counters that report the cycles that are executed on each core. The progression of the counters over time, along with the maximum number of possible cycles per core, can be used to derive CPU utilization.

In order to keep the instrumentation overhead as low as possible, we opt to utilize eBPF instrumentation. This allows us to gain fine-grained control over the sampling frequency and efficient profiling inside the kernel space. To leverage eBPF instrumentation, we built a profiling tool named "trac[5]".

Trac allows to be attached onto a root process. This root process and any process invoked by it are subsequently instrumented for either CPU cycles, resident memory, disk I/O, or network I/O. The gathering of those metrics happens inside the kernel space, where they are collected in a datastructure, called a BPF map. After profiling, these maps can be accessed by the user-space counterpart of the profiling tool. The collected data are processed and presented as CSV time series with a resolution of up to $1ms$.

[3] https://github.com/brianfrankcooper/YCSB

[4] https://pgtune.leopard.in.ua/

[5] https://github.com/omi-uulm/trac

*Isolation.* To isolate processes with Cgroups, we leverage the isolation tool "nsJail[6]". NsJail is a Linux process isolation tool that utilizes the Linux namespace subsystem, Cgroup resource limits, and seccomp-bpf syscall filters to achieve process isolation.

In particular, we use the tool's Cgroup capabilities to isolate the stress-ng CPU load generator, as well as the Postgres database.

The stress-ng CPU load generator itself does not utilize other system resources such as memory, disk I/O and network I/O. As a consequence, we do not isolate these between workloads. Moreover, memory, disk I/O, and network I/O utilized by the Postgres database are negligible for the configuration and workload applied.

## 5 EVALUATION

This section systematically discusses the results of the evaluation outlined in the sections before.

### 5.1 Evaluation Environment

The experimental configuration encompasses a pair of physical servers, symmetrically arranged and equipped with identical components. Both servers feature two Intel CPUs, specifically the "Intel(R) Xeon(R) CPU E5-2630 v3", operating at a base clock frequency of 2.40 GHz with 32 cores. Memory associated with these CPUs totals $16 \cdot 16 = 256$ GiB of DDR4 memory clocked at 2133 MHz. The physical storage disk for the database state is a Samsung SM843TN, which exhibits a Input Output Operations Per Second (IOPS) performance of 15, 000 for "random write" operations.

Communication for actual workload between all nodes is separated and facilitated by Mellanox Technologies' Network Interface Card (NIC) from the "MT27800 ConnectX-5" family, capable of a network throughput of 50 Gbit/s.

Figure 4 visualizes the interaction between the pair of physical servers mentioned above. Here YCSB is responsible to generate and control $W_a$ (Postgres) from a remote host. We do so to limit possible interference on $W_a$ by the load generated by YCSB. The latter should not be accounted for as it would act as a "noisy neighbor". $W_d$ generates its own workload and is not externally controlled.

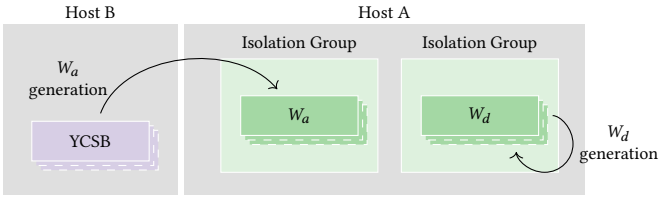



**Figure 4: Workload generation and controlling across hosts**

The complete experiment set-up includes additional auxiliary servers responsible for workflow automation. Notable involved software components are itemized in table 2.

### 5.2 Results

In the following, we iteratively discuss the results of the scenarios presented in table 1. Each scenario is represented by a plot.

[6]https://github.com/google/nsjail

| name | version | note |
|---|---|---|
| Fedora CoreOS | 39 | Operating system version |
| Linux Kernel | 6.5.6 | Kernel used by the operating system Fedora CoreOS |
| k3s | v1.28.4 | Rancher Kubernetes Distribution |
| Argo Workflow | v3.5.4 | The workflow engine to orchestrate experiments and scenarios |
| trac | 0.2.3 | Profiling tool based on eBPF and Aya |
| stress-ng | 0.13.05 | Load generator for CPU |
| YCSB | 0.17.0 | Macro benchmark for databaes |
| Postgres | 15 | SQL based Database management engine |
| nsjail | 3.4 | Process isolation utilizing Linux kernel functions |

**Table 2: Software version list**

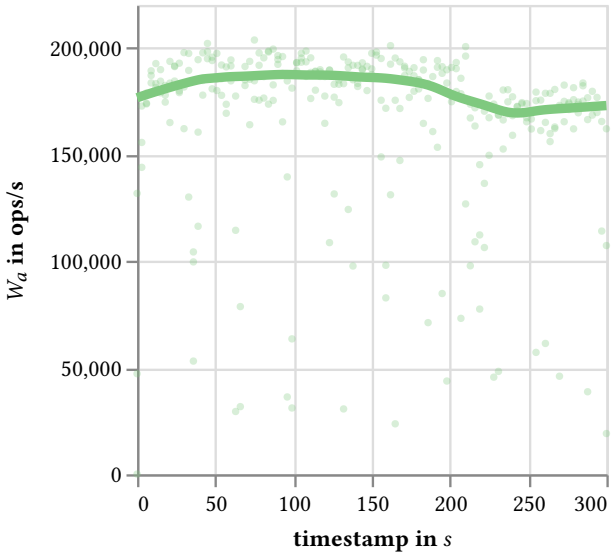



**Figure 5: Baseline scenario**

For the actual isolation metric determination we present an additional graph highlighting the impact on $W_a$ QoS isolation at every observed degree of stress imposed by the disruptive workload $W_d$.

Figure 5 visualizes the baseline scenario. The y-axis shows $W_a$ in operations per second at a given interval in seconds. As described above, this information is provided by YCSB. As each experiment is repeated multiple times and actual operations per second are volatile, we adapted the visualization accordingly. Every measured data point is plotted as a small circle resulting in a scatter plot. An overlay as a smoothed thicker line highlights the trend of those data points. The smoothing algorithm applied implements the Locally Estimated Scatterplot Smoothing (LOESS) method. This results in a trend for this baseline graph that settles roughly at $175,000 ops/s$.

The visualization method for $W_a$ in fig. 6 follows the same principle. However, the visualization of the disruptive workload $W_d$ does not apply said algorithm. Instead, it plots an overlays as the mean

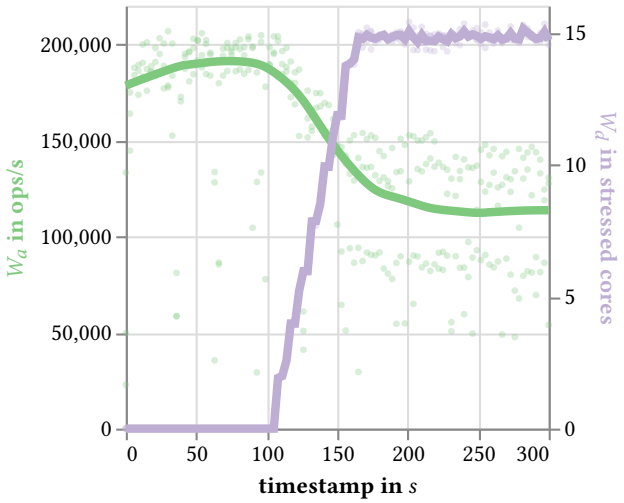



**Figure 6: Harmony scenario**

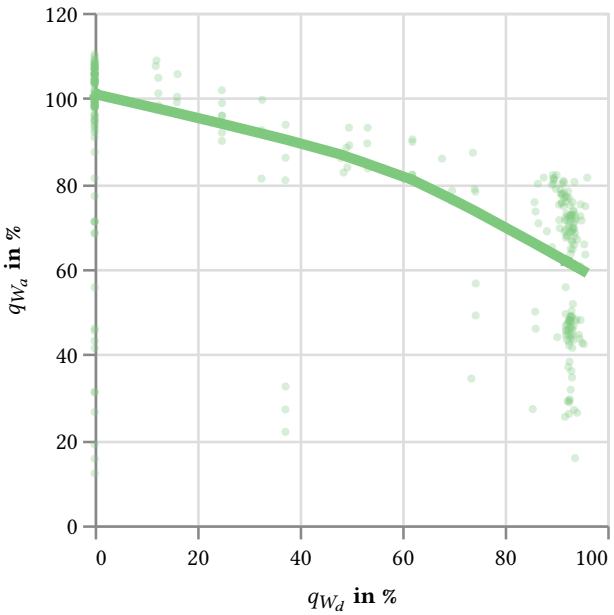



**Figure 7: Isolation impact**

of repeated runs, as these measurements are stable. Thus, the steps of the gradually increasing disruptive workload are easily visible.

Adding this disruptive workload $W_d$ has a significant impact on the behavior of $W_a$. After an initial pausing duration of 100$s$ we can see an immediate degradation of $ops/s$ for $W_a$. This gets worse as $W_d$ reaches its full utilization and results in a degradation of $W_a$ of almost 50%.

Most importantly, neither workload ever exceeds 50% of the physical system capacity, as defined by its assigned CPU cycle limit. This means that the CPU cycles isolation works well considering the fact that no workload is able to exceed its limit. This is in direct conflict of the 50% QoS degradation observed. It is evident that a harmonic split of the seemingly available total resource of CPU cycles can have an impact on each other's CPU performance.

A more detailed visualization with a specific focus on the impact on isolation is presented in fig. 7. Here, the x- and y-axes represent the relative degradation ratio of the workload as defined in section 2.4 with the dimension of time completely removed. Therefore, this graph represents $q_{W_a}$ for every $q_{W_d}$. Again, because of the volatile nature of the measure points, we present the graph as a trend overlay over a scatter plot. Here, we can see a slight change in the degree of degradation above 50% of $q_{W_d}$. What is also easily visible here is that good isolation between $W_a$ and $W_d$ is represented by a higher value, while worse isolation is represented by a lower value within the interval of $[0, 1]$. In table 3 discrete interesting values of fig. 7 are presented. Taking into account the equidistant $q_{W_d}$ values in the interval $[0.1, 0.9]$ of this table results in $\mathbf{I_{avg} = 0.83}$ for eq. (5). Furthermore, we can also calculate $\mathbf{I_{max} = 0.60}$. These values are naturally different from each other, as they both describe different properties of the isolation function $\rho$.

The observations above lead to the following interpretation.

***Interpretation.*** The reason behind the observation that $W_d$ can have such a huge impact on $W_a$ even though they should not impact each other can be manifold. However, two aspects seem to play an important role here.

**Table 3: Isolation metrics comparison**

| $q_{W_d}$ in % | $q_{W_a}$ in % |
|---|---|
| 0.0 | 93.6 |
| 10.0 | 92.0 |
| 20.0 | 89.7 |
| 30.0 | 87.5 |
| 40.0 | 86.0 |
| 50.0 | 84.7 |
| 60.0 | 82.6 |
| 70.0 | 78.8 |
| 80.0 | 72.2 |
| 90.0 | 62.5 |
| 95.0 | 56.8 |

The system we execute our experiment on features two hyperthreading enabled CPUs. Theoretically speaking, they are able to fully utilize all logical cores with maximum cycles if the workload fits. This was observed in previous work of the authors [20]. However, the workload in terms of QoS decreases significantly when the actual physical cores are fully utilized. This assumption is indicated by the slight change of slope in fig. 7 at $q_{W_d} \approx 50\%$. Although more cycles could be utilized by the respective workloads while staying within their cycle limit, they are not able to use them to maintain their QoS.

Another limiting factor could be due to the saturation of only loosely related resources in regard to CPU cycles. This could be due to the overhead induced by process scheduling. Thus, CPU cycles

could be increasingly reserved for such essential tasks, leading to even more starvation of $W_a$.

## 6 RELATED WORK

A prevalent method for assessing the isolation capability involves calculating the $I_{plr}$ as outlined in eq. (1). In line with this approach, previous studies commonly determine this on a per-resource basis [12, 18, 22, 23]. We extend these findings with considerations regarding QoS.

Silva et al. reviewed the effectiveness of resource isolation for QoS isolation in the past [16]. They state that providing QoS for application performance requires more than just guaranteeing a certain allocation of CPU, memory, or I/O resources. We support their findings for the more recent Cgroups v2 and extend them with further measurements and an isolation quantification model.

## 7 CONCLUSION

Over the course of this work, we designed and implemented a sophisticated experimental setup that allowed us to execute two workloads against each other in order to measure their isolation from each other. We have deliberately chosen a very specific scenario, where a synthetic "abiding" database under constant workload competes against a "disruptive" stressor that utilizes the CPU as high as possible.

We determine that those two workloads influence each other even when their CPU limits are evenly shared across the available resources without any overbooking. Neither workload exceeds its limit, but the impact on the QoS of the abiding database is clearly visible.

As a consequence, we can see that mere CPU isolation is insufficient for more complex workloads outside of micro-benchmarks that try to escape them. Aspects like hyper-threading and CPU scheduling overhead are CPU related resources that are not isolated as probably expected. Applying these findings to real-world scenarios requires in situ system tests to determine the actual impact on QoS when co-locating tenants.

The results presented in this work can be considered as a first preliminary step towards more effective QoS isolation. From this point on we see various possible future directions. One is the configuration of stricter isolation environments with limited hyper-threading and possibly system call filtering mechanisms of sandboxes. Another direction could be the improvement of instrumentation to pinpoint the actual saturated resource resulting in a drop in QoS. Lastly, those considerations could be repeated for other Cgroup, different isolation technologies or other workloads.

## REFERENCES


[1] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. ACM, Indianapolis Indiana USA, 143–154. https://doi.org/10.1145/1807128.1807152

[2] Jörg Domaschka, Simon Volpert, Kevin Maier, Georg Eisenhart, and Daniel Seybold. 2023. Using eBPF for Database Workload Tracing: An Explorative Study. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. ACM, Coimbra Portugal, 311–317. https://doi.org/10.1145/3578245.3584313

[3] Chris Down. 2021. 5 Years of Cgroup v2: The Future of Linux Resource Control. USENIX Association.

[4] CSRC Content Editor. [n. d.]. Quality of Service (QoS) - Glossary | CSRC. https://csrc.nist.gov/glossary/term/quality_of_service.

[5] Siqian Gong, Beibei Yin, Zheng Zheng, and Kai-Yuan Cai. 2019. Adaptive Multivariable Control for Multiple Resource Allocation of Service-Based Systems in Cloud Computing. *IEEE Access* 7 (2019), 13817–13831. https://doi.org/10.1109/ACCESS.2019.2894188

[6] Brendan Gregg. 2017. Linux eBPF Tracing Tools. https://www.brendangregg.com/ebpf.html.

[7] Leila Helali and Mohamed Nazih Omri. 2021. A Survey of Data Center Consolidation in Cloud Computing Systems. *Computer Science Review* 39 (Feb. 2021), 100366. https://doi.org/10.1016/j.cosrev.2021.100366

[8] Tejun Heo, J Weiner, V Davydov, L Thorvalds, P Parav, T Klauser, S Hallyn, and K Khlebnikov. 2015. Control Group V2. https://www.kernel.org/doc/Documentation/admin-guide/cgroup-v2.rst.

[9] Samuel Kounev, Klaus-Dieter Lange, and Jóakim von Kistowski. 2020. *Systems Benchmarking: For Scientists and Engineers*. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-41705-5

[10] Rouven Krebs, Christof Momm, and Samuel Kounev. 2012. Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. In *Proceedings of the 8th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA '12)*. Association for Computing Machinery, New York, NY, USA, 91–100. https://doi.org/10.1145/2304696.2304713

[11] Clive Longbottom. 2017. *The Evolution of Cloud Computing: How to Plan for Change*. BCS Learning & Development Ltd, Swindon, UK.

[12] Jeanna Neefe Matthews, Wenjin Hu, Madhujith Hapuarachchi, Todd Deshane, Demetrios Dimatos, Gary Hamilton, Michael McCabe, and James Owens. 2007. Quantifying the Performance Isolation Properties of Virtualization Systems. In *Proceedings of the 2007 Workshop on Experimental Computer Science - ExpCS '07*. ACM Press, San Diego, California, 6–es. https://doi.org/10.1145/1281700.1281706

[13] Steven McCanne and Van Jacobson. 1993. The BSD Packet Filter: A New Architecture for User-Level Packet Capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings (USENIX'93)*. USENIX Association, USA, 2.

[14] Cristian Ruiz, Emmanuel Jeanvoine, and Lucas Nussbaum. 2015. Performance Evaluation of Containers for HPC. In *Euro-Par 2015: Parallel Processing Workshops (Lecture Notes in Computer Science)*, Sascha Hunold, Alexandru Costan, Domingo Giménez, Alexandru Iosup, Laura Ricci, María Engracia Gómez Requena, Vittorio Scarano, Ana Lucia Varbanescu, Stephen L. Scott, Stefan Lankes, Josef Weidendorfer, and Michael Alexander (Eds.). Springer International Publishing, Cham, 813–824. https://doi.org/10.1007/978-3-319-27308-2_65

[15] Daniel Seybold and Jörg Domaschka. [n. d.]. PostgreSQL - Configuration Tuning. https://benchant.com/en/blog/postgresql-configuration-tuning.

[16] Marcio Silva, Kyung Dong Ryu, and Dilma Da Silva. 2012. VM Performance Isolation to Support QoS in Cloud. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, Shanghai, China, 1144–1151. https://doi.org/10.1109/IPDPSW.2012.140

[17] Ali Sunyaev and Ali Sunyaev. 2020. Cloud computing. *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies* (2020), 195–236.

[18] Xuehai Tang, Zhang Zhang, Min Wang, Yifang Wang, Qingqing Feng, and Jizhong Han. 2014. Performance Evaluation of Light-Weighted Virtualization for PaaS in Clouds. In *Algorithms and Architectures for Parallel Processing (Lecture Notes in Computer Science)*, Xian-he Sun, Wenyu Qu, Ivan Stojmenovic, Wanlei Zhou, Zhiyang Li, Hua Guo, Geyong Min, Tingting Yang, Yulei Wu, and Lei Liu (Eds.). Springer International Publishing, Cham, 415–428. https://doi.org/10.1007/978-3-319-11197-1_32

[19] Theodore Ts'o. [n. d.]. *Event Tracing — The Linux Kernel documentation*. https://docs.kernel.org/trace/events.html Accessed: 2023-02-27.

[20] Simon Volpert, Benjamin Erb, Georg Eisenhart, Daniel Seybold, Stefan Wesner, and Jörg Domaschka. 2023. A Methodology and Framework to Determine the Isolation Capabilities of Virtualisation Technologies. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*. ACM, Coimbra Portugal, 149–160. https://doi.org/10.1145/3578244.3583728

[21] Simon Volpert, Sascha Winkelhofer, Stefan Wesner, and Jörg Domaschka. 2024. An Empirical Analysis of Common OCI Runtimes' Performance Isolation Capabilities. In *Proceedings of the 2024 ACM/SPEC International Conference on Performance Engineering*. ACM, London United Kingdom. https://doi.org/10.1145/3629526.3645044

[22] Xingyu Wang, Junzhao Du, and Hui Liu. 2022. Performance and Isolation Analysis of RunC, gVisor and Kata Containers Runtimes. *Cluster Computing* (Jan. 2022). https://doi.org/10.1007/s10586-021-03517-8

[23] Miguel G. Xavier, Israel C. De Oliveira, Fabio D. Rossi, Robson D. Dos Passos, Kassiano J. Matteussi, and Cesar A.F. De Rose. 2015. A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 253–260. https://doi.org/10.1109/PDP.2015.67

# Self-Service Performance Testing Platform for Autonomous Development Teams

Aleksei Vasilevskii
Performance & Observability Team
Wolt
Munich, Germany
aleksei.vasilevskii@wolt.com

Oleksandr Kachur
Performance & Observability Team
Wolt
Helsinki, Finland
alexander.kachur@gmail.com

## ABSTRACT

In the modern fast paced and highly autonomous software development teams, it's crucial to maintain a sustainable approach to all performance engineering activites, including performance testing. The high degree of autonomy often results in teams building their own frameworks that are not used consistently and may be abandoned due to lack of support or integration with existing infrastructure, processes and tools.

To address these challenges, we present a self-service performance testing platform based on open-source software, that supports distributed load generation, historical results storage and a notification system to trigger alerts in Slack messenger. In addition, it integrates with GitHub Actions to enable developers running load tests as part of their CI/CD pipelines.

We'd like to share some of the technical solutions and the details of the decision-making process behind the performance testing platform in a scale-up environment, our experience in building this platform and, most importantly, rolling it out to autonomous development teams and onboarding them into the continuous performance improvement process.

## CCS CONCEPTS

• **Software and its engineering → Software performance**.

## KEYWORDS

performance testing, continuous integration, microservices, autonomous teams

**ACM Reference Format:**
Aleksei Vasilevskii and Oleksandr Kachur. 2024. Self-Service Performance Testing Platform for Autonomous Development Teams. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3652268

## 1 INTRODUCTION

Like many other software organizations developing cloud-native applications, we at Wolt have chosen microservices architecture because of the high degree of autonomy in software engineering teams, that enables feature development and short time to production [14]. The downside is that as the size of the organization grows, it becomes exponentially more difficult to maintain operational visibility over a system consisting of hundreds of microservices [5, 19]. This applies not only to functional aspects, but also to non-functional requirements such as performance and scalability. In a traditional organization, having a centralized performance team responsible for these properties may still work, but in a dynamic scale-up setup, where workloads are constantly increasing and team staffing is lagging behind, the only viable solution is to distribute the responsibility to the development teams [5]. However, this decision can lead to divergent approaches and tools for performance testing, with each team implementing its own framework that works best for its particular use case, but may be far from the "global organizational optimum" [20]. In addition, the groundwork associated with performance testing can be seen as "less important" by teams under pressure to develop features, resulting in poor integration with existing infrastructure and processes [5, 19]. The potential for such artifacts to be reused by other teams, or to be collaborated on and pushed toward a standardized platform, remains relatively low, exacerbating silo problems caused by team communication overhead [5, 6, 21].

Given these constraints, the scope of a dedicated performance team is shifting more and more toward standardized tools, development experience, knowledge sharing, and the definition of best practices. Clearly, there are challenges to building a standardized performance testing platform, but there are also benefits: reducing engineering effort by streamlined teams, eliminating code duplication, empowering teams with the necessary tools and knowledge, and accelerating time to market [21]. Continuous performance testing integrated into CI/CD processes enables shorter feedback loops for software changes. Ongoing platform support, common tooling, and new feature development all increase team buy-in, making performance testing a standard practice rather than an obscure one-off activity.

## 2 REQUIREMENTS

The primary purpose of the performance testing platform is to provide a unified way to conduct repeatable end-to-end system performance tests on existing cloud-native infrastructure. The need to implement an in-house platform stems from the fact that there is no tool-agnostic open-source implementation that meets our requirements. Existing platforms are often product- or company-specific [12], making it impossible to reuse them in a different environment. Other solutions, however, may be too generic [11] and require the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652268

implementation of missing components and integrations, which can be as time-consuming as building a solution from scratch. More specifically, the high-level requirements in our case were:

(1) Cloud-native, cost-effective and scalable solution.
(2) HTTP, WebSocket and gRPC traffic generation.
(3) Easily extensible and configurable.
(4) Support both manual tests and integrate into existing CI pipelines.
(5) Automation for test execution and results evaluation.
(6) Long-term results storage with possibility to visualize trends.
(7) Results repeatability.

When it came to a build vs. buy decision, it was clear that no existing SaaS solution could meet our needs in a cost-effective way. On the other hand, building a tool from scratch wasn't a viable option: different teams were already using different tools, and the longer it would take to release the first version of the framework, the harder it would be to migrate existing tools [5].

There are many decent open-source performance testing tools to integrate with, and this presents the next challenge: choosing the right tool that meets our needs, but is also suitable for most engineering teams [3]. Wolt is a multi-language environment with the main programming languages being Python, Scala and Kotlin, which makes it impossible to force a tool with a specific language (e.g. Locust), while tools that lack one (e.g. ab) or are mostly GUI based (e.g. JMeter) and do not provide the required level of flexibility. Another hard requirement is load generation efficiency at high throughput levels, reaching up to 10k RPS. In addition, the tool of choice should provide decent real-time reporting that is easy to understand and use for comparison, integration capabilities for CI/CD pipelines, observability tools and Kubernetes, and extensibility.

Considering all of the above requirements, Gatling seemed like the optimal choice, see Table 1. Most of the engineers were already familiar with one of the JVM languages, and the learning curve for mastering a framework with expressive DSL built on top of Scala was relatively flat. On the technical side, Gatling convinced us with reasonable load generation efficiency, measurement precision, extensibility, and a wide range of supported protocols [8].

During the early development stages of our performance testing platform, we noticed that test results were showing significant variations that can be attributed to the cloud environment itself [9, 10]. Since we wanted to measure the actual performance characteristics of our services in shared Kubernetes clusters, the only way to improve the repeatability of the measurements was to increase the minimum test duration to collect enough data for a statistically meaningful measurement during each execution [18].

## 3 ARCHITECTURE

We now present the high-level architecture of our solution. As shown in Figure 1, it consists of a Gatling-based load generation application (wolt-load-test), a results storage and analysis module (Witness), custom GitHub Actions to trigger tests from CI/CD pipelines and Argo Workflows to schedule tests on Kubernetes.

The load test can be triggered either automatically by calling the corresponding GitHub Action in a CI/CD pipeline or manually by a user via the Argo WebUI or CLI. In both cases, an Argo Workflow is triggered to start a Kubernetes job with the wolt-load-test workload

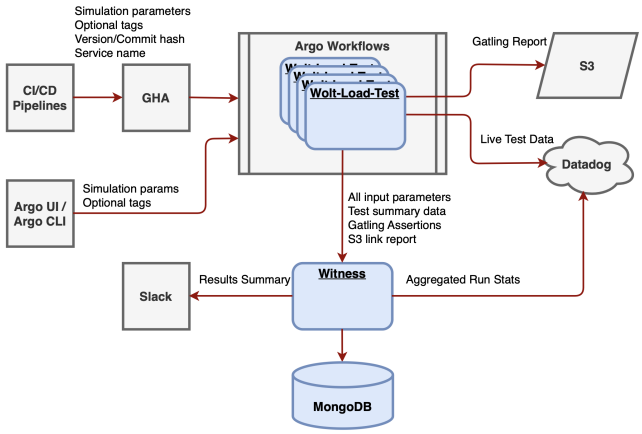



**Figure 1: High-level architecture.**

in the target cluster [13]. The wolt-load-test uses a custom reporter module to send live data to Datadog via the Statsd protocol [7]. Once a load test is complete, the framework generates a report, persists it to an AWS S3 bucket, and sends a summary of the test run to our custom results management module, Witness.

Witness stores the summary of the test run in MongoDB, sends a notification to a predefined Slack channel, and provides a REST API that can be consumed for further automated processing.

### 3.1 Load Generation

We use a custom wrapper around the open-source Gatling tool for load generation, which reduces the learning curve for developers and provides features for a smoother development experience such as standard service-to-service authentication, integration with observability tools (e.g. logs, metrics, traces), and test data management. At its core, the framework provides load test definition templates with multiple levels of hierarchy that can be easily combined. At the top of this hierarchy is a Simulation that defines load levels and durations across one or more Scenarios. Each Scenario combines Requests from one or more Services into a sequential execution chain. A Service contains basic configuration elements such as target host, name, and description, and bundles Requests for ease of navigation. Finally, a Request combines request templates with test data providers - Feeders [8].

In addition to standard Gatling feeders, we also implemented custom Redis-based feeders that support more advanced data structures such as hashmaps and that can be used to store shared or TTLed data. This effectively reduced the amount of memory required for wolt-load-tests with extremely large datasets, and enabled the handling of auto-expiring JWT tokens used for authentication.

The wolt-load-test has its own repository and a CI/CD pipeline that packages all load simulations in an executable JAR, creates a Docker image around this JAR and publishes it to an AWS ECR repository for deployment to respective environments. The load test

dispatch is handled by Argo Workflows, an open source container-native workflow engine for orchestrating parallel jobs on Kubernetes implemented as a Kubernetes CRD [1]. This engine was already in place for scheduling other types of jobs, so it was natural to reuse the existing infrastructure.

Our primary observability platform is Datadog, but Gatling does not integrate with it out of the box. To get around this limitation and avoid the overhead of maintaining a fork of Gatling, we created custom runtime monitoring for load tests by implementing a Statsd protocol reporter using the Byte Buddy framework. Byte Buddy is a code generation and manipulation library for creating and modifying Java classes during the runtime of a Java application and without the help of a compiler, which is widely used in Java Agents of APM and observability tools [23]. To instrument the Gatling code and make the instrumentation as lightweight as possible, we created our own Java Agent. The agent was included in the manifest of the executable JAR in the Launcher-Agent-Class to be launched before the main method of the application is invoked, see Listing 1.

**Listing 1: Maven configuration for loading Agent.**

```
<transformers>
  <transformer implementation="ManifestResourceTransformer">
      <manifestEntries>
          <Main-Class>io.gatling.app.Gatling</Main-Class>
          <Premain-Class>com.wolt.Agent</Premain-Class>
          <Can-Retransform-Classes>true</Can-Retransform-Classes>
          <Launcher-Agent-Class>com.wolt.Agent</Launcher-Agent-Class>
      </manifestEntries>
  </transformer>
</transformers>
```

In this configuration, the JVM first attempts to invoke the agentmain method on the agent class, as shown in Listing 2.

**Listing 2: Agentmain implementation.**

```
public static void agentmain(String args, Instrumentation inst) {
    ByteBuddyAgent.install();
    new AgentBuilder.Default()
            .with(AgentBuilder.Listener.StreamWriting.toSystemOut()
            .withTransformationsOnly())
            .type(named(Transformers.STATS_ENGINE.label))
            .transform(Transformers.STATS_ENGINE.transformer)
            .type(named(Transformers.RESULT_PROCESSOR.label))
            .transform(Transformers.RESULT_PROCESSOR.transformer)
            .installOn(inst);
```

Our agent uses Byte Buddy Advice to transform the Gatling classes and inject the custom code for Statsd monitoring. The high-level workflow of the agent is shown as a sequence diagram in Figure 2.

With this agent in place, we were able to send real-time test data to Datadog and provide an integrated developer experience for both load test results and system-under-test metrics, traces, and profiling data in one place. A dashboard with live test data is shown in Figure 3.

The results we got with the custom agent were quite promising, and we found a few more uses for it. In our reporting pipeline, we relied on Gatling reports, which contain statistics calculated over the entire duration of a load test, including a ramp-up and a steady load period. In some cases, e.g. when running tests in CI/CD pipelines, it's extremely beneficial to calculate statistics over measurements taken only in the second phase, during the steady

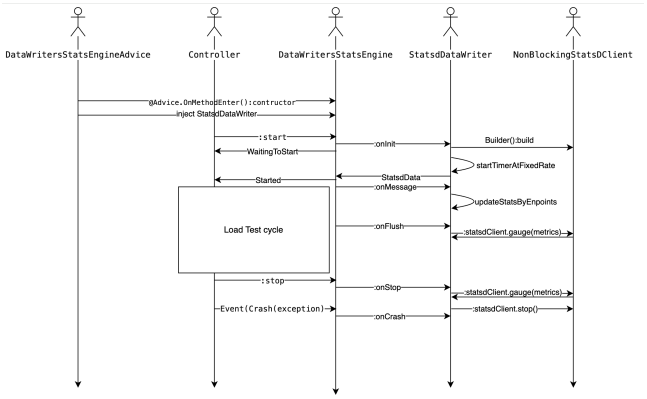



**Figure 2: Sequence diagram for custom runtime monitoring.**

load period. This filters out the cold-start effects that occur in recently deployed applications, and the measurement results are more accurate, reliable, and repeatable. We initially considered developing a custom simulation log processor, but this would have meant rewriting the logic already present in Gatling and introducing a potentially recurring maintenance effort. Instead, we decided to write another ByteBuddy Advice that would skip all data before a specified timestamp during the native Gatling report processing step, see Listing 3.

**Listing 3: Advice for skipping ramp-up period.**

```
public class ResultHolderAdvice {
  @Advice.OnMethodEnter(skipOn = Advice.OnNonDefaultValue.class)
  public static boolean skipRecordProcessing(
    @Advice.FieldValue("minTimestamp") Long startTime,
    @Advice.Argument(value = 0, readOnly = false) Object o) {

    if (o instanceof RequestRecord r) return r.start() < startTime;
    if (o instanceof GroupRecord r) return r.start() < startTime;
    if (o instanceof ErrorRecord r) return r.timestamp() < startTime;
    if (o instanceof UserRecord r) return r.timestamp() < startTime;
    return true;
  }
}
```

We found another useful application for the Byte Buddy Agent. At the end of each test, wolt-load-test reports a summary of the results to our storage service - Witness. Instead of developing and maintaining separate scripts that are not part of the main codebase, we decided to implement yet another Advice attached to io.gatling.charts.report.ReportsGenerator that makes an appropriate HTTP call to Witness once the report data is available.

## 3.2 Test Execution

In our setup, load tests are run as pods in a shared Kubernetes cluster. This allows us to simulate service-to-service traffic in the exact same environment used by our applications, including concurrency of resource allocation, possible network shenanigans, etc. [19]. At the same time, we benefit from using existing infrastructure for configuration and secret management, resources provisioning, and generate no additional maintenance overhead. The downside of this approach is the potential for load generation to interfere with other workloads - or vice versa - the common "noisy neighbor problem"

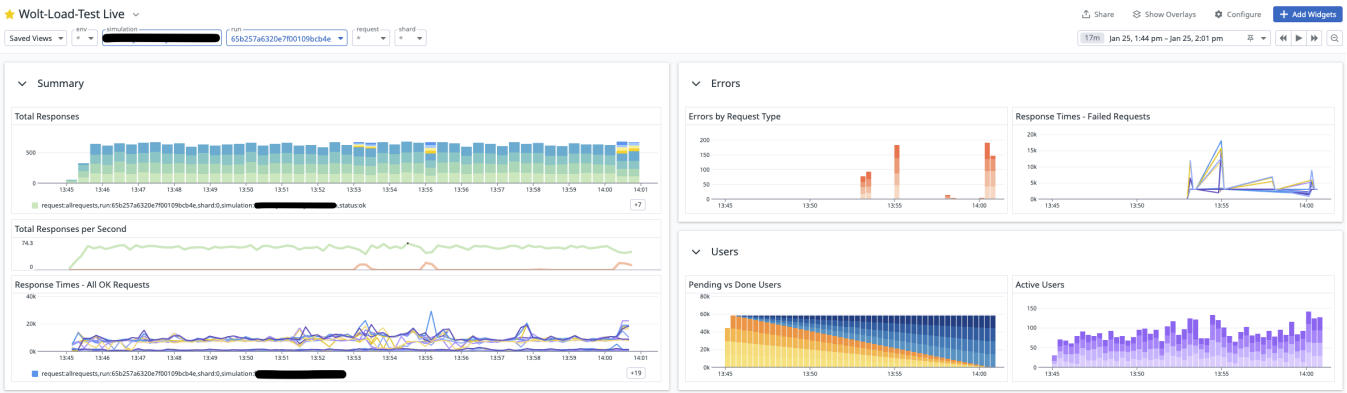



**Figure 3: Wolt-load-test Live Dashboard.**

[24]. This is mitigated by applying anti-affinity rules to specific pods, and by exposing the Kubernetes resource configuration for the wolt-load-test pod as part of the test run configuration. The latter enables a more granular resource management: regular CI load tests get the bare minimum to run, while larger production-sized workloads can allocate much more resources. Actual usage can be monitored using the standard Kubernetes dashboards in Datadog.

One of the alternatives we considered was an external load generation point, such as a separate Kubernetes cluster dedicated to running load test workloads. This would allow us to exercise all components along the external path, including firewalls, external load balancers, etc. However, this would come at a significant cost, both in terms of running an additional Kubernetes cluster and the traffic between the clusters. Given the rate limiters and firewall rules set on the external route, this initiative would require even more effort to invest in workarounds, so it was left out of the scope for now.

Starting a test execution with our solution is possible in several ways. The simplest and most straightforward is to trigger an Argo Workflow via WebUI or CLI by specifying several run parameters, such as simulation class name, maximum target load level, ramp-up and test durations, and some optional custom parameters. It's also possible to specify how many pods should be scheduled for a given workload - we call them shards - and how much resource should be allocated to each shard.

In fact, the original implementation had no sharding support and was quite simple: it just started a single pod with the desired simulation and post-processing step.

After running this setup for some time, we realized that conducting load tests from a single pod can produce results of insufficient quality in terms of connection patterns, load distribution, and resource allocation. Since Argo Workflows also supports more sophisticated DAGs, we split the simple two-step "run test - collect results" job into a multi-step workflow. This new workflow included a preparation step to transform input data and generate a common test run ID, a fan-out step to launch multiple wolt-load-test pods in parallel, and a post-processing step to collect and combine all results into an aggregated test report, as shown in Figure 4. This change not only allowed for more accurate load generation, but also greatly improved the scalability characteristics of our load testing platform, making it suitable for conducting large-scale performance tests.

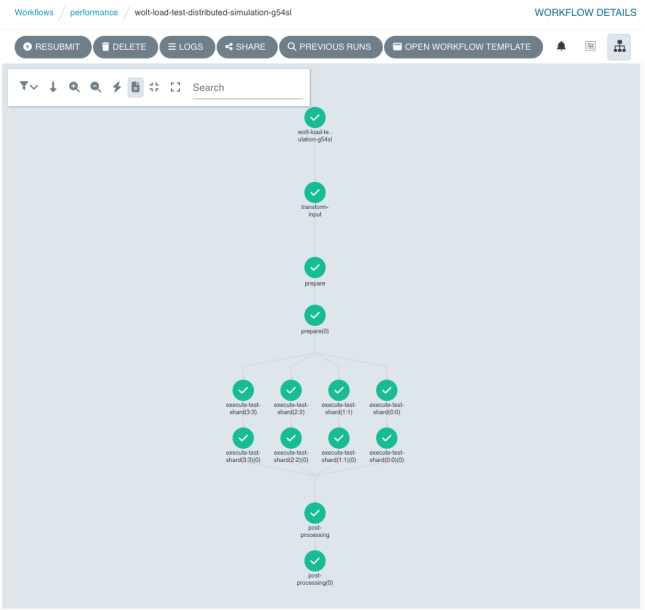



**Figure 4: Argo Workflow for wolt-load-test with multiple shards.**

There is a caveat to the fan-out approach: the start times of the pods may not be perfectly synchronized due to possible scheduling delays in Kubernetes. However, in practice, these delays are usually insignificant and can be ignored for most tests. To further mitigate this, we are considering having a dedicated pool of nodes for running load tests and introducing a synchronization checkpoint during test startup.

Integrating performance testing into services' CI/CD pipelines was as simple as writing a reusable GitHub Action that triggers the appropriate Argo Workflows. This allowed engineering teams to store load test configurations in their respective service-under-test

GitHub repositories and have full control over how and when they run load tests in CI.

A special case is the production environment: in some cases, it's the only environment where meaningful performance measurements can be made due to size, cost, or data volume constraints [19]. In order to support the performance testing in production, we had to implement additional safety measures to avoid any negative impact on the live system, to remain compliant, and to provide auditability of production changes. To this end, the production Argo Workflow instance was hardened to prevent users from running workflows via the WebUI or CLI. Instead, a pull request to a central GitHub repository and a config deployment is required to start any type of load test in production. The downside of this approach, of course, is the reduced user experience compared to the interactive WebUI, but it was accepted by the engineering community as a reasonable compromise.

## 3.3 Data Store and Feedback Loop

Upon completion of the test, a Gatling report is generated by wolt-load-test and stored in an S3 bucket with a unique run ID. Searching and exploring reports directly in the S3 bucket was acceptable in the early stages of adoption with a relatively small number of reports, but proved to be quite cumbersome in the long run. Unfortunately, there are no established open-source results tracking tools that would meet our needs and provide all the integrations we were looking for. So we decided to develop our own service that would simplify the management of test results. To reduce the resource footprint and keep the service minimalistic, we chose Golang as the programming language. Since the concept of test results and the expected access patterns fit well into the document-oriented paradigm, we chose MongoDB for the storage layer.

**Listing 4: Data structure representing a test report.**

```
type GatlingReport struct {
    ID primitive.ObjectID `bson:"_id" json:"id,omitempty"`
    SimulationName string `json:"simulation_name,omitempty"`
    ServiceName string `json:"service_name,omitempty"`
    StartedAt int64 `json:"started_at,omitempty"`
    Version string `json:"version,omitempty"`
    Tag string `json:"tag,omitempty"`
    Params SimulationParams `json:"params,omitempty"`
    Stats GatlingStats `json:"stats,omitempty"`
    Assertions AssertionsResult `json:"assertions,omitempty"`
}
```

The initial Witness implementation included an HTTP REST interface to accept test result summaries with all aggregated per-request and global statistics: response time percentiles, assertions, request and failure rates, along with test metadata that allowed each test run to be uniquely identified [18]. The latter included run ID, simulation class name, load profile details such as max RPS and duration, test start timestamp, service name, version, and tags - see Listing 4. This information is used to group test runs by type for historical comparisons, for example, to distinguish between CI-triggered runs, low-, medium-, and high-load tests, and so on.

Just keeping this information in Witness and providing an HTTP REST interface to read it didn't add much value by itself. Not many people outside of the performance team were using it. To get more traction, we introduced some integrations, the most important of which is a Slack bot: Witness sends a notification to a predefined Slack channel when a test starts, and another message when a test finishes. The latter provides a brief summary and links to relevant information, as shown in Figure 5, which can be used as an initial entry point for detailed analysis based on the metrics, tracing, logs, and profiling information available in Datadog.

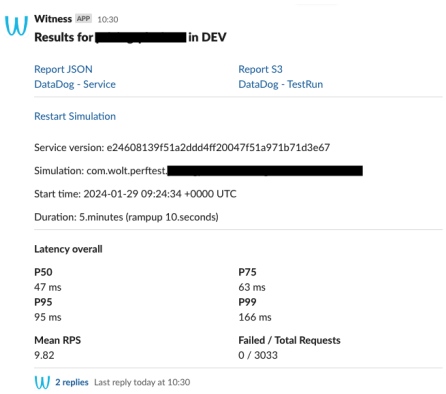



**Figure 5: Witness run summary in Slack.**

This message also provides historical values for that specific combination of simulation class name, load profile, and tag, as well as assertion evaluation results - if any are defined for the simulation. If there are any assertion failures, an additional alert message is generated. This Slack integration has been extremely useful for engineering teams to get an early indication of performance degradation when running load tests as part of their CI/CD pipelines. As a side effect, the Witness Slack channel now acts as a global log of all performance tests, with quick links to all relevant details.

On the other hand, searching for messages in Slack and manually comparing them was not ideal for evaluating long-term trends for specific simulations. To address this, Witness was enhanced with a Statsd integration with Datadog to plot measurement trends.

# 4 ORGANIZATIONAL CHALLENGES

One of the biggest challenges with the self-service performance testing platform was not on the technical side of things, but rather on the organizational side. With multiple development teams having a different vision [5, 6, 19, 21] of how performance testing should look like and what tools should be used, and given the high degree of autonomy in these teams, it was not possible to force any one solution. A much better approach was to first build a foundation by making it as easy and straightforward to use as possible, integrating it with existing processes and infrastructure, and providing the core functionality common to all teams and departments.

In our case, the first step was to make the framework cloud-native, running as a container in Kubernetes with standard infrastructure tools, as opposed to the more common practice of spinning up dedicated EC2 instances for load testing. The next step was to integrate authentication and user management, which had been an integral part of the first release of the platform. With this feature set as a solid foundation, we began to seek out teams that were maintaining their own load tests and interviewed them about their

pain points and desired improvements. By implementing these in our platform, we were able to gradually migrate their homegrown solutions to the common platform. Using this gradual adoption approach and promoting the platform in internal meetings and workshops, wolt-load-test/Witness became the de facto standard for running load tests across the company.

A special case were teams using non-JVM based languages for development, such as Python or Javascript. These teams were understandably reluctant to introduce an entirely new language just for the sake of writing tests. To make it a success story, we provided even more support and guidance to get started, more concrete reference test implementations, and finally, more thorough PR reviews. In the end, the benefits of having great developer experience, tooling support, and a fully integrated solution outweighed the burden of setting up the project and learning the few bits and pieces of Gatling DSL and Scala.

The best part that both product engineering and the performance teams like about the established collaboration is that there's no gatekeeping or heavy reliance on the framework maintainers, since all engineering teams own their load testing code and can run any experiments they want. CI checks on pull requests ensure that the platform is always in a runnable state, and for complex changes or larger feature requests, engineering teams can always get support from the performance team [21].

The rollout of the testing platform was accompanied by a detailed set of guidelines and best practices for systematically addressing performance activities. These guidelines were marketed internally as a "Performance Engineering Framework" with actionable items and tracking for individual teams.

The overall methodology of platform adoption at Wolt can be described in the following steps:

(1) The performance team reaches out to development teams that are not yet onboarded to discuss their needs. This activity can be scheduled on a regular basis for specific teams to increase their engagement. The ultimate goal is to make development teams proactive and ensure that they go through this process on their own, without external requests. Alternatively, development teams can contact the performance team with support requests.
(2) The two teams work through the action items in the "Performance Engineering Framework" and determine if the service should be enrolled for the platform. This depends primarily on the service's APIs, expected load and criticality, as well as any third-party dependencies, data volumes, and overall complexity.
(3) If a service is deemed eligible for enrollment, the two teams evaluate how well it fits into the existing feature set of the platform. If any deficiencies are identified, the performance team proceeds with the implementation of new features.
(4) Once all requirements are met, the development team implements load tests and bindings for integration into their CI/CD process. The entire process is well documented with many examples and detailed step-by-step instructions. If the development team has little or no prior experience with the platform, the performance team can support them with thorough reviews of the changes or draft PRs.
(5) Once the initial implementation is complete, the development team gains full ownership of their load tests and can evolve them as needed. If questions arise during the evaluation of the results, the performance team can assist with root cause analysis.

In rare cases, some services may be excluded in step 2. This usually happens for applications that do not fit well into the current platform paradigm, such as stream processing applications [11]. For these applications, alternative solutions can be considered, such as canary releases or traffic mirroring in production to evaluate performance without impacting real users [4, 22].

Ultimately, addressing these organizational challenges influenced the way engineers approached load testing and helped foster a healthy performance engineering culture, making performance testing a standard part of the development process rather than an obscure one-off exercise. Other notable impacts include freeing up engineering resources by retiring existing scattered testing solutions, improving the efficiency of resource allocation in the cloud and reducing associated costs based on performance testing results, and increasing reliability KPIs in various parts of the Wolt system.

## 5 RELATED WORK

At the time we started developing the platform, there were a few similar end-to-end performance testing tools and publications. However, most of them either target a specific technology or provide a different subset of features.

One example is the MongoDB's Distributed Systems Infrastructure [12]. This framework is used to conduct fully automated performance testing in a CI environment for MongoDB clusters, as well as to provision and deploy the clusters. While it has some similarities to our solution, such as the cloud-native approach and support for both manual and automated CI benchmarks, it can only be used to generate loads for MongoDB and integrates only with the Evergreen CI system [16].

Dell Technology uses JaaS - JMeter as a Service - a performance testing solution built on top of JMeter, Docker, Elastic and Axon - to validate Dell servers before shipping them to customers [17]. It provides distributed load generation for multiple workloads, including HTTP and database traffic, live dashboards and results storage. JaaS does not support running tests natively in Kubernetes and does not provide automated results analysis in the feedback loop.

Theodolite is a framework for benchmarking the scalability of cloud-native applications, running on Kubernetes. It automates the benchmarking process by deploying the system under test (SUT) on a Kubernetes cluster, generating load on the SUT, and collecting performance metrics during load generation [11]. This advanced framework is similar to our solution in many ways, and it provides a set of out-of-the-box benchmarks for streaming processing applications such as Apache Kafka Streams and Apache Flink. For all other types of traffic generation - including HTTP and gRPC - it requires a custom implementation to be provided externally. Test execution is triggered by deploying a custom resource definition (CRD) to a Kubernetes cluster, which is similar to our approach of using Argo Workflows to schedule load generating pods. The distinguishing feature of Theodolite is how it runs isolated experiments for different load intensities and provisioned resources for

the SUT. It provides a set of search strategies to evaluate possible combinations of resources and loads based on configurable service level objectives (SLOs). To store and analyze results, it utilizes a persistent volume, a Grafana server, and a set of Jupyter notebooks.

## 6 FUTURE WORK AND CONCLUSION

The platform we have presented in this submission, is still a work in progress and there are several features and improvements in the works.

One of these improvements touches on the currently used threshold based alerts via assertions: these tend to be flaky, and we have considered using one of the change-point detection algorithms to introduce outlier detection and reduce false positives, e.g. by implementing E-Divisive with Means or similar approaches [15].

In addition, we plan to collect aggregated statistics on application-side metrics such as CPU and memory utilization, profiling and tracing data, etc. and bundle them with the results summary to provide resource utilization comparisons and regression analysis.

Developer experience with the platform can be further enhanced by introducing an interactive integration with Slack, e.g. by providing an easy way to re-run a failed load test directly from the Slack message (party implemented), or by introducing a chat bot functionality to manage load tests without the need for the Argo WebUI or CLI.

Another improvement to the developer experience is planned integration with the internal development portal based on Backstage [2]. This would allow performance measurement data to be embedded into a service health scorecard, to track the progress on "Performance Engineering Framework" action items and provide a single point of entry for all interactions with the platform directly from Backstage.

To address the scheduling delays, completely separate load generation from the system under test, and have a way to stress all components via external endpoints, we had considered setting up an additional Kubernetes cluster dedicated to load testing. However, this would add significant fixed costs and maintenance overhead. At this point, we don't have a specific use case that would justify this effort, but we may revisit this idea in the future. A more efficient solution would be to have a dedicated pool of nodes for running load tests and introduce a synchronization checkpoint.

In this submission, we have presented our solution for a self-service performance testing platform for microservices. This is a scalable, fully integrated performance testing framework built from open-source components by a platform performance engineering team that has been widely adopted in a large engineering organization.

## REFERENCES


[1] Argo. 2018. *Argo Workflows.* https://argo-workflows.readthedocs.io

[2] Backstage. 2020. *What is Backstage?* https://backstage.io/docs/overview/what-is-backstage

[3] Chris Baeckstrom. 2021. *Comparing K6, Gatling and JMeter.* https://www.redline13.com/blog/2022/09/comparing-k6-gatling-and-jmeter/

[4] Jeremy J. Carroll, Pankaj Anand, and David Guo. 2021. Preproduction Deploys: Cloud-Native Integration Testing. arXiv:2110.08588 [cs.NI]

[5] Adrian Cockcroft. 2016. *Microservices workshop.* https://www.slideshare.net/adriancockcroft/microservices-workshop-craft-conference

[6] Melvin E. Conway. 1968. How Do Committees Invent? *Datamation* (April 1968). http://www.melconway.com/research/committees.html

[7] Datadog. 2015. *DogStatsD.* https://docs.datadoghq.com/developers/dogstatsd/

[8] Gatling. 2013. *Gatling.* https://gatling.io/docs/gatling/

[9] Mohammad Hajjat, Ruiqi Liu, Yiyang Chang, T. S. Eugene Ng, and Sanjay Rao. 2015. Application-specific configuration selection in the cloud: Impact of provider policy and potential of systematic testing. In *2015 IEEE Conference on Computer Communications (INFOCOM).* 873–881. https://doi.org/10.1109/INFOCOM.2015.7218458

[10] Sen He, Glenna Manns, John Saunders, Wei Wang, Lori Pollock, and Mary Lou Soffa. 2019. A statistics-based performance testing methodology for cloud applications. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019).* Association for Computing Machinery, New York, NY, USA, 188–199. https://doi.org/10.1145/3338906.3338912

[11] Sören Henning and Wilhelm Hasselbring. 2021. Theodolite: Scalability Benchmarking of Distributed Stream Processing Engines in Microservice Architectures. *Big Data Research* 25 (July 2021), 100209. https://doi.org/10.1016/j.bdr.2021.100209

[12] Henrik Ingo and David Daly. 2020. Automated system performance testing at MongoDB. In *Proceedings of the workshop on Testing Database Systems (SIGMOD-/PODS '20).* ACM. https://doi.org/10.1145/3395032.3395323

[13] Kubernetes. 2020. *Kubernetes Documentation Concepts Workloads Workload Management Jobs.* https://kubernetes.io/docs/concepts/workloads/controllers/job/

[14] James Lewis and Martin Fowler. 2014. *Microservices.* https://martinfowler.com/articles/microservices.html

[15] Mark Leznik, Md Shahriar Iqbal, Igor Trubin, Arne Lochner, Pooyan Jamshidi, and André Bauer. 2022. Change Point Detection for MongoDB Time Series Performance Regression. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering* (Bejing, China) *(ICPE '22).* Association for Computing Machinery, New York, NY, USA, 45–48. https://doi.org/10.1145/3491204.3527488

[16] MongoDB. 2017. *Evergreen.* https://www.mongodb.com/blog/post/testing-linearizability-jepsen-evergreen-call-me-continuously

[17] Vishnu Murty. 2023. Distributed WorkLoad Generator for Performance & Load Testing Using Opensource Technologies. https://raw.githubusercontent.com/ltb2023/ltb2023.github.io/master/slides/LTB23_VMurty.pdf

[18] Alessandro Vittorio Papadopoulos, Laurens Versluis, André Bauer, Nikolas Herbst, Jóakim von Kistowski, Ahmed Ali-Eldin, Cristina L. Abad, José Nelson Amaral, Petr Tůma, and Alexandru Iosup. 2021. Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. *IEEE Transactions on Software Engineering* 47, 8 (2021), 1528–1543. https://doi.org/10.1109/TSE.2019.2927908

[19] Matt Ranney. 2016. *What I Wish I Had Known Before Scaling Uber to 1000 Services.* https://www.youtube.com/watch?v=kb-m2fasdDY

[20] Tanya Reilly. 2022. *The Staff Engineer's Path: A Guide For Individual Contributors Navigating Growth and Change.* O'Reilly Media.

[21] M. Skelton, M. Pais, and R. Malan. 2019. *Team Topologies: Organizing Business and Technology Teams for Fast Flow.* IT Revolution. https://books.google.fi/books?id=oFdRuAEACAAJ

[22] Cindy Sridharan. 2018. *Testing in Production, the safe way.* https://copyconstruct.medium.com/testing-in-production-the-safe-way-18ca102d0ef1

[23] Rafael Winterhalter. 2014. *Why runtime code generation?* https://bytebuddy.net

[24] Ailin Yang. 2022. *"Noisy Neighbors" Problem in Kubernetes.* https://www.intel.com/content/www/us/en/developer/articles/technical/noisy-neighbors-problem-in-kubernetes.html

# Overhead Comparison of Instrumentation Frameworks

David Georg Reichelt
Lancaster University Leipzig
Leipzig, Saxony, Germany

Lubomír Bulej
Charles University
Prague, Czech Republic

Reiner Jung
Kiel University
Kiel, Germany

André van Hoorn
Universität Hamburg
Hamburg, Germany

## ABSTRACT

Application Performance Monitoring (APM) tools are used in the industry to gain insights, identify bottlenecks, and alert to issues related to software performance. The available APM tools generally differ in terms of functionality and licensing, but also in monitoring overhead, which should be minimized due to use in production deployments. One notable source of monitoring overhead is the instrumentation technology, which adds code to the system under test to obtain monitoring data.

Because there are many ways how to instrument applications, we study the overhead of five different instrumentation technologies (AspectJ, ByteBuddy, DiSL, Javassist, and pure source code instrumentation) in the context of the Kieker open-source monitoring framework, using the MooBench benchmark as the system under test. Our experiments reveal that ByteBuddy, DiSL, Javassist, and source instrumentation achieve low monitoring overhead, and are therefore most suitable for achieving generally low overhead in the monitoring of production systems.

However, the lowest overhead may be achieved by different technologies, depending on the configuration and the execution environment (e.g., the JVM implementation or the processor architecture). The overhead may also change due to modifications of the instrumentation technology. Consequently, if having the lowest possible overhead is crucial, it is best to analyze the overhead in concrete scenarios, with specific fractions of monitored methods and in the execution environment that accurately reflects the deployment environment. To this end, our extensions of the Kieker framework and the MooBench benchmark enable repeated assessment of monitoring overhead in different scenarios.

## CCS CONCEPTS

• **General and reference → Performance**; • **Software and its engineering → Software performance**; *Software maintenance tools.*

## KEYWORDS

software performance engineering, benchmarking, performance measurement, monitoring overhead

**ACM Reference Format:**
David Georg Reichelt, Lubomír Bulej, Reiner Jung, and André van Hoorn. 2024. Overhead Comparison of Instrumentation Frameworks. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3652269

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3652269

## 1 INTRODUCTION

Understanding application performance at runtime requires collecting metrics such as response times, resource usage, or error rates during application execution. When applied to production environments, this process is referred to as performance monitoring [19]. While monitoring can be done at different levels, including business, application, and the infrastructure, we focus on the application level in this work.

Because performance monitoring is often orthogonal to features delivering value to application users, it is common to implement monitoring as a cross-cutting concern using instrumentation, i.e., by inserting instructions into software that explicitly create monitoring records associated with relevant performance events. An alternative method for obtaining information about application performance is *sampling*, in which the system periodically collects snapshots of generic (e.g., code location being executed, stack traces, memory consumption) or application-specific metrics (e.g., request queue depth, average response time) exposed through a suitable framework (e.g., JMX beans).

While sampling allows controlling the trade-off between (a fixed) overhead and accuracy by setting the sampling period, instrumentation producing explicit records of relevant events is generally more flexible, because it can produce data usable for both monitoring and tracing. However, the overhead of instrumentation-based monitoring is much more variable and potentially more difficult to control. In this paper, we focus on the baseline overhead due to the use of a particular instrumentation technology.

During instrumentation, monitoring code (*probe*) is inserted into the monitored application to create monitoring records corresponding to various events occurring during application execution. The injection of probes can be achieved using different instrumentation technologies. The instrumentation process, depicted in Figure 1, is generally driven by a application monitoring frameworks, such as OpenTelemetry or Kieker, which also determine the instrumentation technology used to insert probes into system under test.

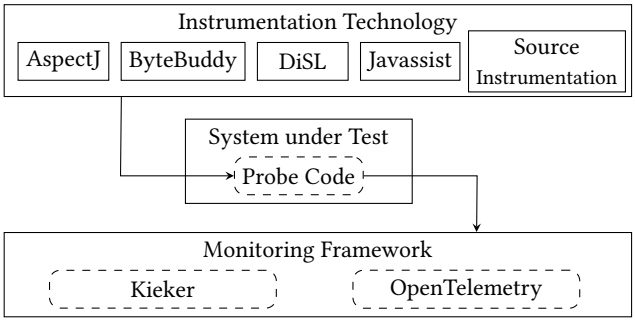



**Figure 1: Instrumentation Process for Monitoring Frameworks**

In this paper, we compare the overhead of different instrumentation technologies using the Kieker [5] monitoring framework. In addition, we compare the measured overhead to the overhead of OpenTelemetry using its default instrumentation technology ByteBuddy.

The instrumentation can be done at compile time, at application start time, using a `javaagent` with a `premain` method, or at application runtime, using an `agentmain` method. In this work, we compare the monitoring overhead that is introduced due to compile-time or application start-time instrumentation. The overhead of dynamic instrumentation has already been examined [6].

By comparing the monitoring overhead of the instrumentation technologies AspectJ[1], ByteBuddy[2], DiSL [11], Javassist[3], and source code instrumentation, we find that (1) There are significant differences using the instrumentation technologies, e.g., using AspectJ creates up to 30 % more overhead than directly instrumenting the source code, (2) source code instrumentation has the lowest overhead in our benchmarking executions, (3) OpenTelemetry has significantly higher overhead, that is not caused by its instrumentation technology ByteBuddy, and (4) all technologies scale linearly with the call tree depth. Furthermore, our extensions of the Kieker framework and the MooBench benchmark enable repeated assessment of monitoring overhead, e.g., in case the underlying JVM or the instrumentation technologies are changed.

This paper is structured as follows. We first provide a brief overview of the instrumentation technologies, the application monitoring framework, and the benchmark used. Then we present our experimental setup and results, which we subsequently compare to related work. Finally, we give a summary and an outlook.

## 2 FOUNDATIONS

In this section, we first provide an overview of the instrumentation technologies that are the subject of this study. We then describe the Kieker monitoring framework used to carry out the comparison and the information collected for monitoring purposes. Finally, we describe the benchmark used as the system under test.

[1]https://eclipse.dev/aspectj/
[2]https://bytebuddy.net/
[3]https://www.javassist.org/

### 2.1 Instrumentation Technologies

In this work, we use five instrumentation technologies: Source instrumentation, AspectJ, ByteBuddy, DiSL, and Javassist. Source instrumentation operates directly on the source code and transforms it into instrumented source code. Subsequent compilation produces instrumented bytecode which is executed by the JVM. Source code instrumentation requires access to the source code and is language-specific, and generally need to be done at application *compile time*. The other technologies employ *bytecode manipulation* and insert instrumentation code directly into the bytecode of application classes, producing instrumented bytecode to be executed by the JVM. This approach is (mostly) language-agnostic and offers increased flexibility compared to source instrumentation.

Frameworks such as AspectJ, ByteBuddy, and Javassist support *compile time* instrumentation, i.e., instrumentation is performed prior to application execution using bytecode that is either downloaded or produced by compilation. An alternative is to perform instrumentation at application *load time*, which completely decouples instrumentation from the build process. This is done using instrumentation API provided by the JVM, usually through a `javaagent` with a `premain` method which triggers bytecode transformation of the classes being loaded by the JVM. This approach is supported by all of the above mentioned frameworks. The different instrumentation processes are summarized in Figure 2.

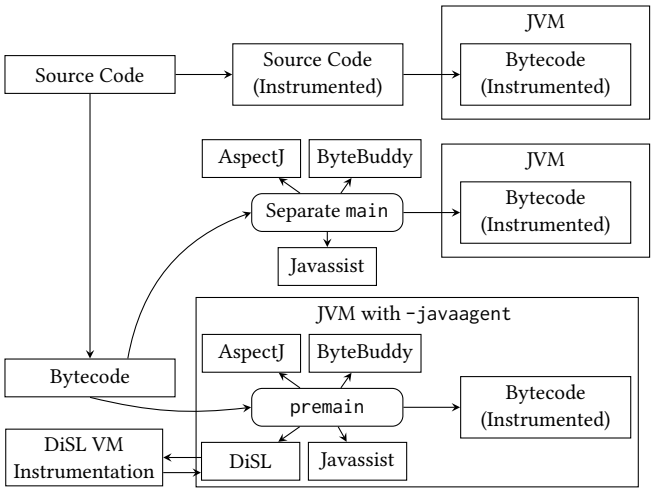



**Figure 2: Instrumentation Options: Source Code, Compile Time and Load Time**

*Source Instrumentation.* The most direct way of injecting monitoring probes is by adding them to the source code of an application. Listing 1 shows this in a simplified way: In the original code of `myMethod`, obtaining the start time, obtaining the end time, and passing this information to a `MonitoringController` is added.

**Listing 1: Manual Instrumentation Example**

```
public void myMethod() {
  long tin = System.nanoTime();
  ....
  long tout = System.nanoTime();
```

```
  MONITORING_CONTROLLER.newOperationExecution(
    tin, tout, "MyClass.myMethod()");
}
```

Manual instrumentation is time-consuming and error-prone, and reduces the maintainability of the source code. Therefore, we developed a tool for automated injection of monitoring probes in Java source code.[4] This reduces the monitoring overhead significantly [16]. While source code instrumentation can reduce the monitoring overhead, it cannot be used in cases where only the bytecode (and not the source code) is available. This includes cases where other programming languages have been used, e.g., Scala or Kotlin. This disadvantage could be fixed by implementing the source instrumentation for other languages.

*AspectJ.* AspectJ was developed to support the modular implementation of crosscutting concerns, i.e., concerns that need to be addressed in different modules. This is called aspect-oriented programming [8]. AspectJ builds on *advices* that consist of *pointcuts* (patterns describing when the pointcut is activated) and advice bodies (the code that should be executed on a matching pointcut). The usability of AspectJ for implementation of domain requirements that are crosscutting concerns is controversial [13]. Nevertheless, its usability for logging and monitoring is indisputable. Internally, AspectJ uses BCEL[5] for low-level adaptation of bytecode.

*ByteBuddy.* ByteBuddy aims to make the adaptation of byte code possible without knowledge of its format. Thereby, it makes it possible to support various use cases of bytecode manipulation for security, logging, or performance monitoring. Especially, the OpenTelemetry API reference implementation relies on ByteBuddy. Internally, AspectJ uses ASM[6] for low-level adaptation of bytecode.

*DiSL.* DiSL[7] (Domain-specific language for instrumentation) was developed primarily for dynamic program analysis [11]. DiSL allows selecting any region of the bytecode for instrumentation, in contrast to AspectJ's model, which only allows the selection of specific points, e.g., operation starts. DiSL also supports synthetic local variables, which facilitate passing of information between advice code in the scope of a single method, e.g., collecting operation start and end times, and passing the operation duration along with context information to the application performance monitoring framework for recording.

Internally, DiSL uses the ASM[8] library that provides a low-level interface for bytecode manipulation. During *load time* instrumentation, DiSL uses a separate VM to execute the instrumentation logic which processes the application classes loaded by the VM in which the application executes. This eliminates perturbations in the application VM caused by the instrumentation logic using common classes that may be subject to instrumentation (due to their use by the application).

*Javassist.* Javassist[9] is a Java bytecode manipulation library [2]. It supports changing the bytecode by the specification of adapted bytecode and source code. By adding a `javagent` and adding a `ClassTransformer` (which is a class from `java.lang`), monitoring libraries can interact with the class loading mechanism. Since bytecode manipulation needs a thorough knowledge of the structure of bytecode itself, libraries like Javassist can ease this process. However, the usage of Javassist also requires the (indirect) definition of bytecode manipulation. Checking the instrumentation source at compile-time (like with AspectJ aspects or DiSL instrumentation definitions) is not possible.

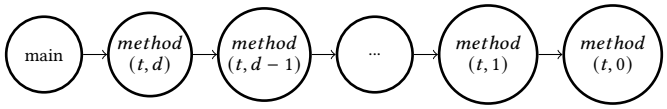



**Figure 3: Call Tree of MooBench, from: [16]**

## 2.2 Kieker

Application performance monitoring (APM) frameworks contain a library and an agent that are responsible for data acquisition.[10] There are both open-source APM tools, such as Kieker [5] and OpenTelemetry[11], and closed-source APM tools, such as Dynatrace APM[12] and the DataDog agent.[13] While their implementation details differ, they are all able to obtain operation execution data, such as operation start and end times, operation signature, and operation's position in the call tree.

Our prior study on the overhead of these frameworks shows that Kieker has the lowest overhead among the open-source tools [15]. Kieker's `OperationExecutionRecord` contains, next to the timing information, the execution operation index (`eoi`) and the execution stack size (`ess`), which make it possible to reconstruct the call tree. In this work, we compare how much overhead different instrumentation technologies cause to collect the data needed to create the `OperationExecutionRecord`.

## 2.3 MooBench

Moobench is a benchmark that compares the overhead of different APM frameworks and their configurations [20]. To measure the overhead for each operation execution, MooBench calls its `monitoredMethod` recursively for a given call tree depth. In the leaf node, a busy waiting is executed, which simulates calculations. The structure of the MooBench call tree is depicted in Figure 3.

Performance measurement is subject to non-determinism because of a number of factors. Some, such as other processes running on the same system or CPU frequency and voltage scaling can be mitigated by platform configuration. Other factors such as differences in memory layout between different runs of the workload (in different processes) need to be addressed by the measurement process. Experiments involving platforms such as the Java or JavaScipt VM need to account for the effects of just-in-time compilation and garbage collection. A rigorous measurement process must collect data from multiple experiment runs, each time using a new (managed language) VM process. Within each run, the measured operation needs to be repeated enough times to get past the warm-up

[4] https://github.com/kieker-monitoring/kieker-source-instrumentation
[5] https://commons.apache.org/proper/commons-bcel/
[6] https://asm.ow2.io/license.html
[7] https://gitlab.ow2.org/disl/disl
[8] https://asm.ow2.io/
[9] https://www.javassist.org/
[10] https://openapm.io/
[11] https://opentelemetry.io/
[12] https://www.dynatrace.com/de/platform/application-performance-monitoring/
[13] https://docs.datadoghq.com/agent/

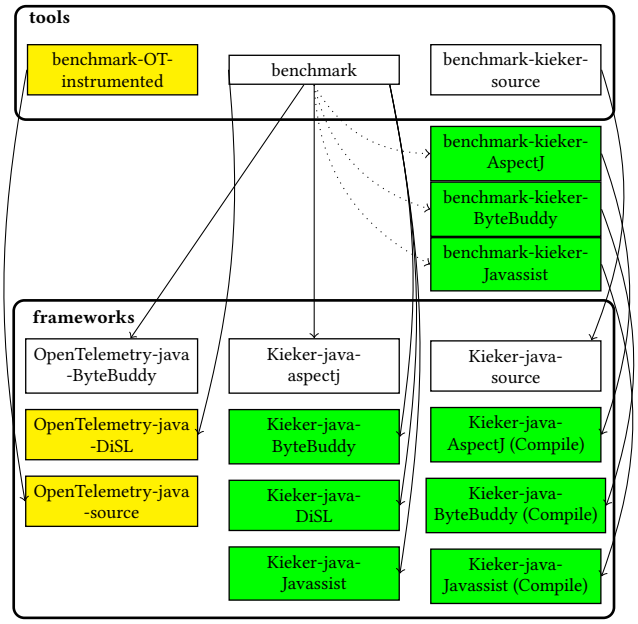



**Figure 4: Architecture of MooBench (Green: Added by us, Yellow: Future extension options)**

phase (dominated by just-in-time compilation) to collect samples representing the expected operation durations. The averages of operation durations obtained from individual runs then provide basis for statistical analysis, e.g., a comparison using the two-sided t-test [4].

MooBench allows comparing different frameworks and execution environments using a rigorous measurement process. Each framework/execution environment combination contains a single folder, e.g., Kieker-java and Kieker-Python with a script to execute measurements. Each script contains configuration parameters for the respective framework, including the name, the command line parameters, and the environment variables.

MooBench also aims to identify the causes of the overhead, assuming that monitoring overhead emerges from instrumentation overhead, data collection, and the writing of data [21]. To this end, MooBench supports performance measurements using multiple configurations: Baseline (without any instrumentation), instrumentation only (with *deactivated monitoring*), monitoring with *no logging* (data is not written to data sink), and full monitoring (writes data to a sink, e.g., binary file, TCP receiver, Zipkin server).

## 3 BENCHMARKING

In this section, we first describe how we implemented the benchmark changes. Afterwards, we describe our configurations for benchmark execution. Based on this, we describe the results of the MooBench performance comparison. Finally, we discuss the scalability of the overhead.

### 3.1 Benchmark Adaptation

To benchmark the different instrumentation technologies, load-time and compile-time instrumentations for each instrumentation technology, the decision of how to handle deactivated monitoring, and the adaptation of the benchmark were necessary. These are described in the following.

*Load-time Instrumentation.* Kieker already supports the AspectJ instrumentation and contains a subproject for automated source instrumentation. Therefore, we additionally implemented probes for ByteBuddy, DiSL, and Javassist. To do so, we started providing separate JARs for each instrumentation (`kieker-bytebuddy`, `kieker-disl`, and `kieker-javassist`), as a `kieker-aspectj`-jar was already provided before.

To create a `javaagent`, the `premain` needs to specify what should be done when the agent is started. AspectJ contains its own `premain` implementation which handles the instrumentation of classes according to joinpoint specification in `aop.xml`. Because ByteBuddy and Javassist require users to implement the instrumentation procedure on their own, we implemented a `javaagent` for each of them. Since the other technologies do not provide a method for joinpoint specification, we use the previously existing Kieker method pattern definition to specify methods that should be instrumented and pass it to the agents through the `KIEKER_SIGNATURES` environment variable. Both agents only support the `OperationExecutionRecord`, as this record is also used for the other MooBench implementations.

*Compile-Time Instrumentation.* Source instrumentation naturally happens at compile-time. To get to know whether the load-time weaving or the bytecode created by the instrumentation technology is causing overhead, we additionally created Kieker `main` methods that instrument an existing JAR using AspectJ, ByteBuddy and Javassist. Afterwards, these are used by specifically tailored benchmark configurations.

*Deactivated Monitoring.* One challenge is the need to support deactivation of monitoring at runtime, either fully or for selected methods. AspectJ and ByteBuddy allow to return directly from the instrumentation methods, and thereby execute the unchanged original method. Because that is not possible with the source instrumentation, we copy the original method body into a branch that is executed when monitoring is disabled or when the monitoring of the method is deactivated. In the case of Javassist and DiSL, we use a separate monitoring class (`OperationExecutionDataGatherer`) that the instrumentation code calls on operation start and operation end. When monitoring is disabled, the operation start invocation returns `null`, indicating to the instrumentation code that the operation end invocation is not necessary. Otherwise it returns a object (`FullOperationStartData`) representing operation data needed to create a monitoring record, which the instrumentation passes to the operation end invocation.

**Listing 2: Exit Handling in DiSL and Javassist**

```
@SyntheticLocal
static FullOperationStartData data;

@Before(marker = BodyMarker.class,
  scope = "MonitoredClass*.*")
public static void beforemain(
  final KiekerStaticContext c) {
  data = OperationExecutionDataGatherer
```

```
    .operationStart(c.operationSignature());
}

@After(marker = BodyMarker.class,
  scope = "MonitoredClass*.*")
public static void aftermain() {
  if (data != null) {
    OperationExecutionDataGatherer
      .operationEnd(data);
  }
}
```

For Javassist, another option is to obtain all monitoring data at the beginning of the method call (including the start time) and to only add the monitoring data if monitoring is enabled. This leads to heavily increased overhead for deactivated monitoring (0.55 $\mu s$, instead of 0.06$\mu s$) and to slightly decreased overhead for enabled monitoring (2.42 $\mu s$ instead of 2.55$\mu s$). This is a good option if all instrumented classes have activated monitoring. If monitoring is deactivated (and might be activated again later), this increases the overhead. Since the other frameworks do not have this option, we did not consider this implementation variant further to have a fair comparison between all frameworks.

*MooBench Adaptation.* Based on these changes, we modified MooBench to support all instrumentation technologies. Before our refactoring, only framework-language combinations were supported, i.e., Kieker-java, Kieker-python, inspectIT-java, and OpenTelemetry-java. After our refactoring, we added the instrumentation technology for every framework, e.g., we renamed the old Kieker-java to Kieker-java-aspectj and additionally implemented Kieker-java-bytebuddy. Since ByteBuddy and Javassist provide their bytecode instrumentation directly, we could reuse most of the code. For DiSL, we needed to call the DiSL starter Python script in every benchmark call. The adapted architecture of MooBench is depicted in Figure 4.

## 3.2 Execution Configuration

After extending the benchmark, we executed it with two environment configurations: An i7-4770 running Ubuntu 22.04 and OpenJDK 11, and a Raspberry Pi 4 running Debian 11 with OpenJDK 11. While using the Raspberry Pi might yield measurement values that are different from typical business application deployments, they have the advantage of being reproducible for other researchers due to their standardization and affordability [9]. For each environment configuration, we ran two experiments: The measurement of the **monitoring overhead** and the measurement of the **overhead scalability**.

For the monitoring overhead, we ran the experiments with MooBench's default parametrization (2 000 000 calls, zero time for busy waiting, so the busy waiting part will only be two calls to `System.nanoTime()`, a call tree depth of 10 and 30 seconds sleep time), but set the number of VM starts to 30 (according to [4], who recommends at least 30 VM starts to gather enough observations for statistical testing).

For the overhead scalability, we also used the default configuration and set the recursion depth to 2, 4, ..., 128 to examine the change of execution time with growing call tree size. To reduce the benchmarking time, we only executed the benchmark on the i7-4770. Our full measurement data are available as a dataset[14].

While executing the experiments, we noticed that the execution duration increased after a certain count of iterations. The effect occurred on every technology. Based on technology and execution environment, the threshold of iteration for this effect to start is between 500 000 and 2 000 000 iterations. After more iterations, the effect becomes stronger. We suspected memory and hard disk writing to be responsible for this effect. Therefore, we tried to change the memory settings (using different configurations between `-Xmx2g` and `-Xmx10g`). Regardless of the memory settings, the effect stayed the same. When reducing hard disk writing by setting a maximum file number that Kieker should write to,[15] we could eliminate this effect. The runtime for this effect is depicted in Figure 5 (with a very strong increase after 5 000 000, and a barely visible effect occurring earlier). Therefore, we set the `maxLogFiles` and will keep this for MooBench's future default configuration.

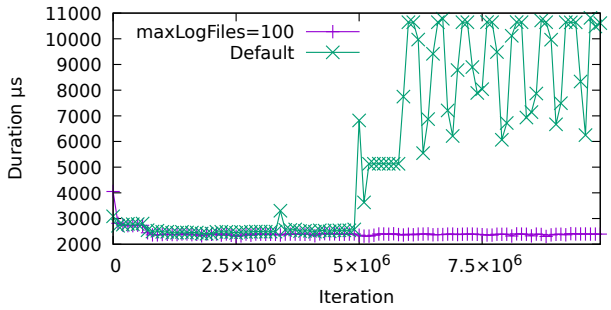



**Figure 5: Average Duration per Iteration With and Without Setting `maxfiles`**

## 3.3 Monitoring Overhead

Table 1 depicts the monitoring overhead for the different technology framework combinations for load-time instrumentation, and Table 2 depicts the monitoring overhead for Kieker for compile-time instrumentation. Overhead in the MooBench context means *execution time caused by monitoring*. Therefore, the measured duration is the measured duration of empty method executions with monitoring. For AspectJ, this would mean full monitoring causes roughly 3.30 $\mu s$ overhead (=3.35$\mu s$ − 0.05$\mu s$) on 10 method calls, indicating an overhead of 0.33 $\mu s$ per node. In a production environment where methods themselves take time, the absolute overhead is expected to stay roughly the same (in the same execution environment), but the relative overhead will be significantly lower. The overhead depends on whether monitoring is deactivated, configured for execution without logging, or fully activated.

*Deactivated Monitoring.* For deactivated monitoring, OpenTelemetry creates a higher overhead than Kieker (at least factor 10). This overhead does not seem to be caused by ByteBuddy, since Kieker's ByteBuddy probe causes much lower overhead. The overhead for

[14] https://doi.org/10.5281/zenodo.10607598
[15] `-Dkieker.monitoring.writer.filesystem.FileWriter.maxLogFiles`

a deactivated AspectJ probe is notably higher than for ByteBuddy, DiSL, Javassist, and source instrumentation (roughly factor 3). ByteBuddy, DiSL, Javassist, and source instrumentation do not have statistically significant differences for compile-time weaving on the i7-4770. On Raspberry Pi, AspectJ compile-time instrumentation is slower than load-time. The remaining technologies differ only slightly.

*No Logging.* In this configuration the monitoring records are stored into a queue that discards them [14]. The performance in this configurations gives an indication how much of the overhead stems from the data collection itself. This configuration is only available for the Kieker probes. In this configuration, AspectJ has a significantly higher overhead than the other instrumentation technologies, and source instrumentation has a significantly lower overhead than all other instrumentation technologies. Surprisingly, we see that ByteBuddy has significantly higher overhead in its compile-time variant than its load-time variant on the Raspberry Pi. Since exactly the same final bytecode is executed, we assume that this is caused by different internal optimizations of the JVM, which won't necessarily occur in a production system.

*Full Monitoring.* In this configuration, the monitoring records are passed on to the application monitoring framework. Compared to OpenTelemetry, Kieker allows collecting traces with a lower overhead. Similarly to *no logging* and *deactivated monitoring* configurations, we see that AspectJ's full instrumentation creates higher overhead than source instrumentation, ByteBuddy, DiSL, and Javassist. As for *no logging*, we see that source instrumentation has the lowest overhead, which comes at the cost of requiring the source code. For the comparison of ByteBuddy, DiSL, and Javassist, we see that their ranking changes depending on whether load-time or compile-time instrumentation is used, and whether execution happens on a the Raspberry Pi or the i7-4700. Therefore, we assume that these differences are caused by different internal optimizations of the JVM, which might be different in production systems.

Looking at these values, we can infer that source instrumentation is the best variant if the source code is available, and that ByteBuddy, DiSL, and Javassist are good candidates for low overhead. Nevertheless, the overhead depends on the execution infrastructure, indicating that comparisons of instrumentation technologies for different software might differ.

It is also notable that the values measured for the baseline configuration are nearly the same as in our previous experiments [15], but the values for the full monitoring configuration changed (i7-4470 Kieker: Mean was 3.4, OpenTelemetry: 6.8). Since Kieker did not have significant changes in its code base, we assume that this is caused by optimizations that happened either in used libraries or in the execution environment, including the JVM (which we updated from OpenJDK 8 to OpenJDK 11) and the Linux Kernel.

| Benchmark | Mean | Standard Deviation | Mean | Standard Deviation |
|---|---|---|---|---|
| | i7-4770 | | Raspberry Pi | |
| Baseline | 0.05 | 0.00 | 0.16 | 0.00 |
| Kieker-java | | | | |
| -aspectj (deactivated) | 0.21 | 0.00 | 0.93 | 0.01 |
| -aspectj (nologging) | 1.68 | 0.02 | 5.63 | 0.12 |
| -aspectj (full) | 3.35 | 0.07 | 12.69 | 2.21 |
| -bytebuddy (deactivated) | 0.07 | 0.00 | 0.31 | 0.01 |
| -bytebuddy (nologging) | 1.08 | 0.01 | 3.42 | 0.09 |
| -bytebuddy (full) | 2.61 | 0.13 | 8.10 | 0.47 |
| -disl (deactivated) | 0.07 | 0.00 | 0.31 | 0.02 |
| -disl (nologging) | 1.21 | 0.01 | 3.94 | 0.11 |
| -disl (full) | 2.75 | 0.18 | 8.30 | 0.67 |
| -javassist (deactivated) | 0.06 | 0.00 | 0.27 | 0.01 |
| -javassist (nologging) | 1.16 | 0.01 | 4.01 | 0.14 |
| -javassist (full) | 2.58 | 0.17 | 8.25 | 0.58 |
| OpenTelemetry-java | | | | |
| -bytebuddy (deactivated) | 3.28 | 0.15 | 14.29 | 0.52 |
| -bytebuddy (full) | 4.98 | 0.18 | 22.41 | 1.13 |

**Table 1: Monitoring Overhead for Load Time Technologies (in $\mu s$)**

| Benchmark | Mean | Standard Deviation | Mean | Standard Deviation |
|---|---|---|---|---|
| | i7-4770 | | Raspberry Pi | |
| Baseline | 0.05 | 0.00 | 0.16 | 0.01 |
| Kieker-java | | | | |
| -aspectj (deactivated) | 0.22 | 0.00 | 1.17 | 0.05 |
| -aspectj (nologging) | 1.66 | 0.02 | 5.88 | 0.19 |
| -aspectj (full) | 3.35 | 0.12 | 13.75 | 3.63 |
| -bytebuddy (deactivated) | 0.06 | 0.00 | 0.27 | 0.01 |
| -bytebuddy (nologging) | 1.17 | 0.02 | 3.83 | 0.17 |
| -bytebuddy (full) | 2.53 | 0.12 | 8.44 | 0.89 |
| -javassist (deactivated) | 0.06 | 0.00 | 0.27 | 0.01 |
| -javassist (nologging) | 1.17 | 0.01 | 3.84 | 0.13 |
| -javassist (full) | 2.50 | 0.06 | 8.19 | 0.58 |
| -sourceinstrumentation | | | | |
| (deactivated) | 0.07 | 0.00 | 0.28 | 0.01 |
| (nologging) | 1.04 | 0.01 | 3.30 | 0.11 |
| (full) | 2.32 | 0.15 | 7.20 | 0.55 |

**Table 2: Monitoring Overhead for Compile Time Technologies (in $\mu s$)**

## 3.4 Overhead Scalability

Figure 6 shows the overhead evolution with increasing call tree depth. It shows a nearly linear increase, which indicates that no instrumentation technology causes serious problems such as memory leaks. We observe that OpenTelemetry (with its ByteBuddy-based probes) has a significantly higher overhead. With Kieker, we observe that AspectJ is slower for all tested call tree depths. For source instrumentation, we see a slightly smaller standard deviation and slightly lower average duration than the bytecode instrumentation technologies.

For performance monitoring, warmup performance is also relevant, as a user might want to detect performance anomalies early in the runtime of an application. Figure 7 shows the evolution of the warmup performance. Here, the steady-state performance is

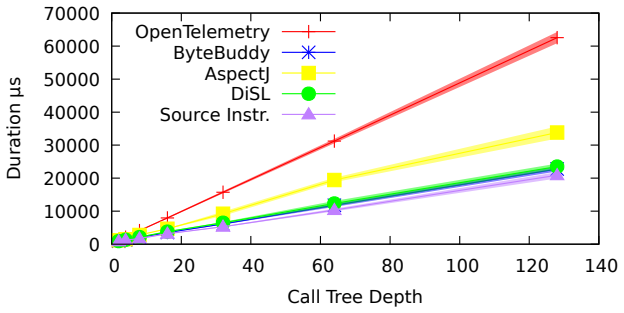



**Figure 6: Steady State Performance on i7-4770**

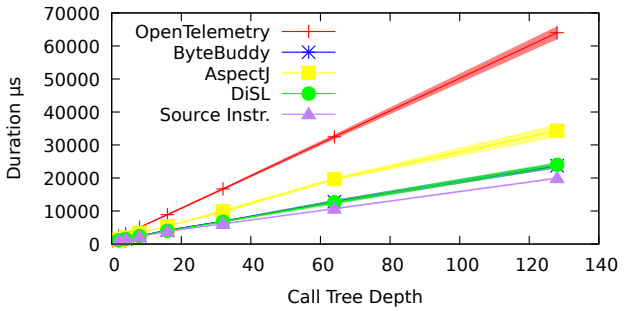



**Figure 7: Warmup Performance on i7-4770**

mainly repeated, with even lower differences between ByteBuddy, DiSL, and Javassist.

## 4 RELATED WORK

Related work to the overhead analysis of instrumentation technologies for APM exists in three fields: Analysis of the monitoring overhead of APM tools, analysis of the overhead instrumentation itself, and analysis of overhead of instrumentation for different infrastructures.

### 4.1 Overhead of APM Tools

Performance and regression benchmarking are a widespread practice for monitoring tools. OpenTelemetry[16] and the monitoring tool GlowRoot[17] provide own benchmarks for their tools. However, comparisons of monitoring tools or of their instrumentation technologies rarely exist. ByteBuddy itself provides a comparison of ByteBuddy, cglib, Javassist, and Java proxies in their basic tutorial.[18] They find that based on different use cases, different instrumentation technologies are faster. Therefore, it is necessary to compare the instrumentation technologies in the context of their usage in monitoring, as we did in this work.

Okanovic et al. [12] already examined the use of AspectJ and DiSL on Kieker. While they pursued the same goal, they did not persistently implement their changes into the Kieker code. Additionally, they used a self-implemented benchmark instead of MooBench and did not examine how manual instrumentation performs in comparison to AspectJ and DiSL.

Banda analyzed OpenTelemetry's performance overhead,[19] which resulted in a commit measuring the span processing overhead.[20] In his work, he compared the performance of different queue types (e.g., `ArrayBlockingQueue` and `ConcurrentLinkedQueue`) with different configurations. He did this only for the exporting part, i.e., he created spans inside of a JMH benchmark. Therefore, in contrast to this work, he focused on the factor of the data processing, whereas we use a benchmark that includes the data creation, data processing, and writing, and focus on the instrumentation part.

Shatnavi et al. [17] examine the monitoring overhead for three human resources and financial management systems, including their web UIs. The systems are built on GWT-Spring, and the instrumentation is done using OpenTelemetry. They evaluate the overhead of a baseline and two usage scenarios of their systems. In their setup, they do not find an overhead of the frontend agent as the frontend is executed at the users' site. Furthermore, they find an overhead of about 3 %-4 % at one backend component and an unacceptable overhead in another component; therefore, they decide to exclude instrumentation for serialization parts of the application.

### 4.2 Overhead of Instrumentation

Chukri et al. [18] present the BISM (Bytecode-Level Instrumentation for Software Monitoring) tool that allows, like DiSL, to instrument source code. They compare the monitoring overhead with AspectJ and DiSL using the DaCapo benchmark suite and find that BISM creates a lower overhead than both of them. Since the tool is not publicly available, a comparison to BISM is not possible.

Horký et al. [7] use DiSL to obtain performance measurements of performance unit tests. While they also apply DiSL for performance measurement, they focus on the evaluation of performance unit test measurements and do not evaluate the overhead caused by the instrumentation.

Horký et al. [6] examine the monitoring overhead by dynamic instrumentation, with dynamic instrumentation implemented using DiSL. They assume that the monitoring overhead emerges from probe presence, the (byte)code manipulation, the optimization, and the data storage. This is different from our approach since they inject probes dynamically, while Kieker's probes are in the code all the time and are enabled or disabled through variable values. Furthermore, they use the SPEC-jbb2015 benchmark instead of MooBench. Additionally, for their measurement, they consider the recursive call as one method call and only send one method call record, whereas Kieker considers each invocation of recursive calls as a single method call record and sends these as single `OperationExecutionRecord`. Finally, they find that dynamic instrumentation using dynamic injection of probes is a promising and overhead-reducing approach for dynamic monitoring. Since they only used DiSL as instrumentation technology, one further work would be to use the instrumentation technologies we used in

[16] For example https://opentelemetry.io/blog/2023/perf-testing/ — OpenTelemetry has a huge suite of benchmarks for different languages.
[17] https://glowroot.org/overhead.html
[18] https://bytebuddy.net/#/tutorial
[19] https://doordash.engineering/2021/04/07/optimizing-opentelemetrys-span-processor/
[20] https://github.com/open-telemetry/opentelemetry-java/commit/23ce8fe3929d98aaaa63ab8d5d7ab2b99dcea85b

this work. Thereby, it could be checked whether instrumentation overhead can be reduced further, e.g., by using Javassist.

### 4.3 Overhead of Instrumentation for Different Infrastructures

Bruening et al. [1] build DynamoRIO, which enables instrumentation at the processor operation level. They compare it to Intel's Pin tool that also allows to instrument applications at the processor operation level. Using SPEC CPU2006, they find that DynamoRIO has an overhead of 21 %, whereas Pin has an overhead of 11 %.

Wasabi is an instrumentation technology for Web Assembly (wasm) [10]. Lehmann and Pradel define it and examine its overhead using the PolyBench benchmark suite that was originally developed for benchmarking wasm itself. Based on the instrumented call, they find that overhead might vary from 2 % up to factor 163 when instrumenting all assembler commands.

For Python, Eghbali and Pradel define DynaPyt for program analysis [3]. They find that the overhead varies between 20 % and factor 16.

All experiments on other languages were done using predefined benchmarks, whereas we used a benchmark specifically suited for monitoring overhead. Therefore, the results are not directly comparable. One possible future work would be to apply the probes we created to JVM benchmarks in order to examine the overhead.

## 5 SUMMARY

In this work, we extended the MooBench benchmark to compare different instrumentation technologies. We compared five instrumentation technologies for the instrumentation framework Kieker: AspectJ, ByteBuddy, DiSL, Javassist, and source instrumentation. We found that AspectJ creates the highest overhead for all configurations, and source instrumentation creates lower overhead in most configurations. For ByteBuddy, DiSL, and Javassist, one or another is faster, depending on the configuration.

In our experiments, we focused on instrumentation that can be enabled and disabled at runtime, thus making adaptive monitoring possible. A static instrumentation that decides on program startup which methods should be instrumented typically has lower overhead. In use cases where this is possible, static probes should be used to obtain minimal overhead.

Reducing the overhead of instrumentation remains a challenge since available technologies change, and, therefore, the most efficient instrumentation technology might change. In future work, we plan to execute our measurements on more heterogeneous hardware and repeat the measurements with future versions of underlying technologies, e.g., newer JVMs or JVMs from different vendors.

Furthermore, MooBench's current implementation focuses on the overhead in terms of CPU time consumption to measure the method execution duration. An extension for other aspects of the overhead (e.g., throughput, CPU, and/or memory usage) and the overhead of measuring different things (e.g., the overhead of jersey for HTTP request processing) would be an important extension in order to minimize performance measurement overhead overall.

Besides monitoring, there are other use cases for instrumentation technologies, including data serialization, logging, mocking, and testing. In this work, we focused on performance for monitoring overhead and, therefore, measured the overhead for changing monitored methods. For other use cases, other aspects of instrumentation, like class creation, might be more important. Therefore, promising future work is also the creation of an instrumentation technology benchmark covering other use cases of instrumentation.

**Acknowledgement** This work is supported by the German Research Foundation (DFG): Project "SustainKieker" (HO 5721/4-1 and HA 2038/11-1).

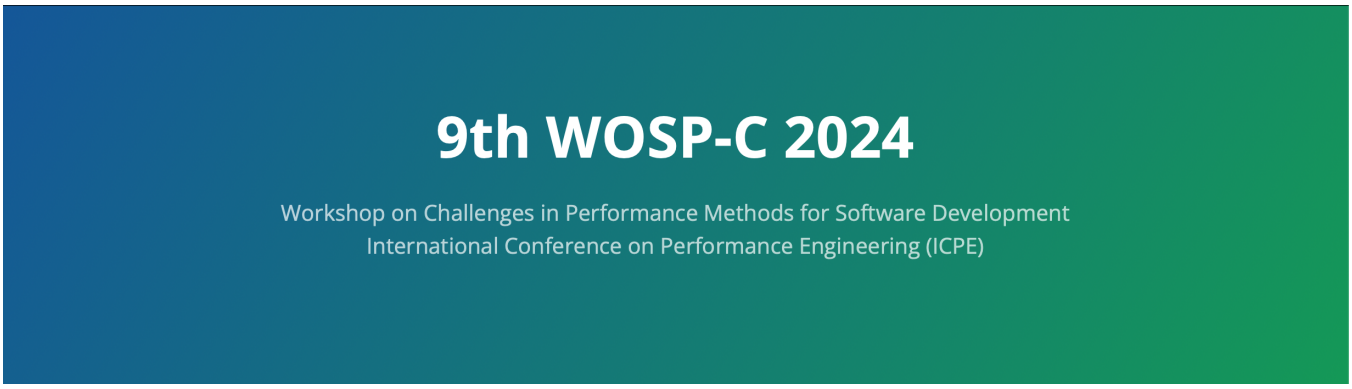
## REFERENCES

[1] Derek Bruening, Qin Zhao, and Saman Amarasinghe. 2012. Transparent dynamic instrumentation. In *Proc. 8th ACM SIGPLAN/SIGOPS Conf. on Virtual Execution Environments*. 133–144.
[2] Shigeru Chiba and Muga Nishizawa. 2003. An easy-to-use toolkit for efficient Java bytecode translators. In *Proc. Int. Conf. on Generative Programming and Component Engineering*. Springer, 364–376.
[3] Aryaz Eghbali and Michael Pradel. 2022. DynaPyt: A dynamic analysis framework for Python. In *Proc. 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 760–771.
[4] Andy Georges, Dries Buytaert, and Lieven Eeckhout. 2007. Statistically Rigorous Java Performance Evaluation. *ACM SIGPLAN Notices* 42, 10 (2007), 57–76.
[5] Wilhelm Hasselbring and André van Hoorn. 2020. Kieker: A monitoring framework for software engineering research. *Software Impacts* 5 (2020), 100019.
[6] Vojtěch Horký, Jaroslav Kotrč, Peter Libič, and Petr Tůma. 2016. Analysis of Overhead in Dynamic Java Performance Monitoring. In *Proc. 7th ACM/SPEC Int. Conf. on Performance Engineering*. ACM, 275–286.
[7] Vojtěch Horký, Peter Libič, Lukáš Marek, Antonin Steinhauser, and Petr Tůma. 2015. Utilizing Performance Unit Tests To Increase Performance Awareness. In *Proc. 6th ACM/SPEC Int. Conf. on Performance Engineering*. ACM, 289–300.
[8] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G Griswold. 2001. An overview of AspectJ. In *Proc. 15th European Conf. on Object-Oriented Programming*. Springer, 327–354.
[9] Holger Knoche and Holger Eichelberger. 2018. Using the Raspberry Pi and Docker for Replicable Performance Experiments: Experience Paper. In *Proc. 8th ACM/SPEC Int. Conf. on Performance Engineering*. 305–316.
[10] Daniel Lehmann and Michael Pradel. 2019. Wasabi: A framework for dynamically analyzing webassembly. In *Proc. 24th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*. 1045–1058.
[11] Lukáš Marek, Alex Villazón, Yudi Zheng, Danilo Ansaloni, Walter Binder, and Zhengwei Qi. 2012. DiSL: A domain-specific language for bytecode instrumentation. In *Proc. 11th Int. Conf. on Aspect-oriented Software Development*. 239–250.
[12] Dušan Okanović, Milan Vidaković, and Zora Konjović. 2013. Towards performance monitoring overhead reduction. In *Proc. 11th IEEE Int. Symposium on Intelligent Systems and Informatics*. 135–140.
[13] Adam Przybyłek. 2018. An empirical study on the impact of AspectJ on software evolvability. *Empirical Software Engineering* 23, 4 (2018), 2018–2050.
[14] David Georg Reichelt, Reiner Jung, and André van Hoorn. 2023. More is Less in Kieker? The Paradox of No Logging Being Slower Than Logging. In *Proc. 14th Symposium on Software Performance*.
[15] David Georg Reichelt, Stefan Kühne, and Wilhelm Hasselbring. 2021. Overhead Comparison of OpenTelemetry, inspectIT and Kieker. In *Proc. 12th Symposium on Software Performance*.
[16] David Georg Reichelt, Stefan Kühne, and Wilhelm Hasselbring. 2023. Towards Solving the Challenge of Minimal Overhead Monitoring. In *Comp. 14th ACM/SPEC Int. Conf. on Performance Engineering*. 381–388.
[17] Anas Shatnawi, Bachar Rima, Zakarea Alshara, Gabriel Darbord, Abdelhak-Djamel Seriai, and Christophe Bortolaso. 2023. Telemetry of Legacy Web Applications: An Industrial Case Study. (2023). https://hal.science/hal-04344518/document
[18] Chukri Soueidi, Marius Monnier, and Yliès Falcone. 2023. Efficient and expressive bytecode-level instrumentation for Java programs. *International Journal on Software Tools for Technology Transfer* 25, 4 (2023), 453–479.
[19] Jan Waller. 2015. *Performance Benchmarking of Application Monitoring Frameworks*. Ph. D. Dissertation. Kiel University, Germany.
[20] Jan Waller, Nils Christian Ehmke, and Wilhelm Hasselbring. 2015. Including Performance Benchmarks into Continuous Integration to Enable DevOps. *ACM SIGSOFT Software Engineering Notes* 40, 2 (3 2015), 1–4.
[21] J. Waller and W. Hasselbring. 2012. A Comparison of the Influence of Different Multi-Core Processors on the Runtime Overhead for Application-Level Monitoring. In *Proc. Int. Conf. on Multicore Software Engineering, Performance, and Tools*. Springer, 42–53.

# 9th Workshop on Challenges in Performance Methods for Software Development: WOSP-C'24 Chairs' Welcome

Luca Traini
luca.traini@univaq.it
University of L'Aquila
Department of Information Engineering, Computer Science, and Mathematics
L'Aquila, Italy

Heng Li
heng.li@polymtl.ca
Polytechnique Montréal
Computer and Software Engineering
Montréal, Canada




## ABSTRACT

We are pleased to welcome you to the 9th Workshop on Challenges in Performance Methods for Software Development – WOSP-C 2024 (https://wosp-c.github.io/wosp-c-24/). This year's edition continues its tradition of being the forum for discussions on novel and unresolved challenges in the field of software performance engineering. The primary areas of interest for the workshop remain consistent with previous editions, but the scope of this year's edition has been extended to include additional novel topics. WOSP-C 2024 will cover topics such as performance assurance processes, the interplay between performance and privacy, green software, performance modeling, and simulation, among others.

We warmly welcome attendees to attend our keynote, and research talks:

- [Keynote] *25+ years of software performance: from integrated system modeling to ML-based analysis, what's next?* Vittorio Cortellessa ( University of L'Aquila).
- [Keynote] *Closing the Loop: Building Self-Adaptive Software for Continuous Performance Engineering*. Marin Litoiu (York University).
- [Work-in-Progress Paper] *Green Software Metrics*. Andreas Brunnert.
- [Short Paper] *Privacy-Preserving Sharing of Data Analytics Runtime Metrics for Performance Modeling*. Jonathan Will, Dominik Scheinert, Seraphin Zunzer, Jan Bode, Cedric Kring and Lauritz Thamsen.
- [Full Paper] *HetSim: A Simulator for Task-based Scheduling on Heterogeneous Hardware*. Marcel Lütke Dreimann, Birte Friesel and Olaf Spinczyk.
- [Full Paper] *Approximating Fork-Join Systems via Mixed Model Transformations*. Rares Dobre, Zifeng Niu and Giuliano Casale.
- [Full Paper] *Establish a Performance Engineering Culture in Organizations*. Josef Mayrhofer.

Putting together WOSP-C 2024 was a team effort. We first thank the authors for providing the content of the program. We are grateful to the program committee, who worked very hard to review papers and provide feedback to the author.s Finally, we thank ICPE for hosting our workshop.

We hope that you will find this program interesting and thought-provoking and that the symposium will provide you with a valuable opportunity to share ideas with other researchers and practitioners from institutions around the world.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651438

**ACM Reference Format:**
Luca Traini and Heng Li. 2024. 9th Workshop on Challenges in Performance Methods for Software Development: WOSP-C'24 Chairs' Welcome. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3629527.3651438

# Closing the Loop: Building Self-Adaptive Software for Continuous Performance Engineering

Marin Litoiu
Electrical Engineering and Computer Science Department, School of IT
York University
Totonto, Ontario, Canada
mlitoiu@yorku.ca

## ABSTRACT

Cloud computing and cloud-native platforms have rendered runtime environments more malleable. Simultaneously, the growing demand for flexible and agile software applications and services has driven the emergence of self-adaptive architectures. These architectures, in turn, facilitate software performance modeling, tuning, optimization, and scaling in a continuous manner, blurring the boundary between development-time and run-time. Self-adaptive software employs feedback loop controllers inspired by control theory or variations of the Monitoring-Analysis-Planning-Acting (MAPE) architecture. Whether implemented in a centralized or decentralized manner, most controllers utilize performance models that are learned or tuned at run-time. This shift implies that software is designed to be observable and controllable during execution, presupposing the co-design of software applications and their runtime controllers.

This talk commences with a succinct overview of the evolution of self-adaptive software, accentuating key milestones along the journey. Subsequently, recent advancements in software performance modeling at runtime and the role of learning-enabled performance management during software operation are presented.

Two recent works are highlighted: one focusing on constructing robust performance models to sustain continuous operation and deployment of cloud-native software, and the other on utilizing multimodal models for performance anomaly detection. The former supports cloud operations like continuous deployment of co-located applications, migration, consolidation of services, or scaling in response to workloads or interferences. The latter is tailored to support performance anomaly detection, localization, and identification of root causes, facilitating swift remediation of faults using generative AI. The final segment of the talk delves into current challenges in developing self-adaptive systems, presenting insights from a recent survey on the state of self-adaptive software in the industry and the challenges perceived by practitioners.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright is held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652910

## CCS CONCEPTS

•Software and its engineering •Software and its engineering~Software organization and properties~Extra-functional properties~Software performance

## KEYWORDS

Self-adaptive software systems, self-optimization, software performance, performance models, cloud computing, machine learning, generative AI

**ACM Reference format:**

Marin Litoiu. 2024. Closing the Loop: Building Self-Adaptive Software for Continuous Performance Engineering, Keynote, In *WOSP-C 2024, Proceedings of ICPE '24 Companion, May 7–11, 2024,London, United Kingdom, 2 pages, https://doi.org/10.1145/3629527.3652910*

## BIOGRAPHY

Marin Litoiu is a Professor of Software Engineering in the Department of Electrical Engineering and Computer Science and in the School of Information Technology, York University. He is also a Fellow of the Canadian Academy of Engineering. Dr. Litoiu leads the Adaptive Software Research Lab and focuses on making large software systems more versatile, resilient, energy-efficient, self-healing and self-optimizing. His research won many awards including the IBM Canada CAS Research Project of the Year Award, the IBM CAS Faculty Fellow of the Year Award for his "impact on IBM people, processes and technology," three Best Paper Awards and two Most Influential Paper Awards. Prior to joining York University, Dr. Litoiu was a Research Staff member with the Centre for Advanced Studies in the IBM Toronto Lab where he led the research programs in software engineering and autonomic computing. He received the Canada NSERC Synergy Award for Innovation in recognition for these collaborative university/industry activities. He was also

recipient of the IBM Outstanding Technical Contribution Award for his research vision on Cloud Computing. Dr. Litoiu is one of the founders of the SEAMS Symposium series—ACM/IEEE Software Engineering for Adaptive and Self-Managing Systems. Dr. Litoiu is also the Scientific Director of "Dependable Internet of Things Applications (DITA)," an NSERC CREATE program.

## ACKNOWLEDGMENTS

This work has been supported by IBM Centre for Advances Studies, National Science and Engineering Research Council of Canada and Ontario Research Fund- Research Excellence.

## REFERENCES

[1] Sarda Komal, Namrud Zakeya, Rouf Raphael, Ahuja Harit, Rasolroveicy Mohammadreza, Litoiu Marin, Shwartz Larisa, and Watts Ian. 2023. ADARMA Auto-Detection and Auto-Remediation of Microservice Anomalies by Leveraging Large Language Models. In Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering (CASCON '23). IBM Corp., USA, 200–205.

[2] T. Zheng, C. M. Woodside and M. Litoiu, "Performance Model Estimation and Tracking Using Optimal Filters," in *IEEE Transactions on Software Engineering*, vol. 34, no. 3, pp. 391-406, May-June 2008, doi: 10.1109/TSE.2008.30.

[3] Y. Rouf, J. Mukherjee and M. Litoiu, "Towards a Robust On-line Performance Model Identification for Change Impact Prediction," *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Melbourne, Australia, 2023, pp. 68-78, doi: 10.1109/SEAMS59076.2023.00018.

[4] D. Weyns, M.Litoiu *et al.*, "Towards Better Adaptive Systems by Combining MAPE, Control Theory, and Machine Learning," *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Madrid, Spain, 2021, pp. 217-223, doi: 10.1109/SEAMS51251.2021.00036.

[5] D. Weyns, M. Litoiu et al.. 2023. Self-Adaptation in Industry: A Survey. ACM Trans. Auton. Adapt. Syst. 18, 2, Article 5 (June 2023), 44 pages. https://doi.org/10.1145/3589227

[6] R. Rouf, M. Rasolveicy, M Litoiu et al., InstantOps: A Joint Approach to System Failure Prediction and Root Cause Identification in Microservices Cloud-Native
Applications, Proceedings of ACM/SPEC ICPE 2024, London, UK, May 2024.

[7] K. Sarda, Z. Narud, M. Litoiu et al. KubePlaybook: A Repository of Ansible Playbooks for Kubernetes Auto-Remediation with LLMs, Proceedings of ACM/SPEC ICPE 2024, London, UK, May 2024.

[8] de Lemos, R. *et al.* (2013). Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds) Software Engineering for Self-Adaptive Systems II. Lecture Notes in Computer Science, vol 7475. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35813-5_1

# 25+ years of Software Performance: From Integrated System Modelling to ML-based Analysis, What's Next?

Vittorio Cortellessa
vittorio.cortellessa@univaq.it
University of L'Aquila
Italy

## ABSTRACT

A new era has been opened at the end of last century in the performance analysis research area, when an explicit and independent role has started to be given to software in performance analysis of computing systems. Indeed, software has moved from being a monolithic element, strictly dependent on the platform where it is deployed and exclusively aimed at producing values to parameterize a platform model, to become an independent model itself, with its own components and interactions. This change has impacted all fields of this research area, such as: modeling languages, processes for analysis and synthesis of software models, platform model parameterization, performance model solution techniques, interpretation of results, benchmarking and performance testing. It has also represented one of the triggers that lead to the birth of a research community around the computing system performance issues strictly related to software aspects. Indeed, in 1998 the first ACM Workshop on Software and Performance (WOSP) took place, with the aim of getting together researchers and practitioners of software area with the ones of the performance area, so to offer a playground where different skills and expertise could join and originate a new vision on the role of software in performance assessment. This talk attempts to reconstruct the road of software performance research that has started at the time of the first WOSP event in 1998 down to today events (i.e., ICPE conference, WOSP-C and other workshops). The spirit of the talk is to observe the evolution of this research area, including successful and (apparently) unsuccessful directions. Some promising directions will be tentatively sketched by "standing on the shoulders of giants".

## CCS CONCEPTS

• **Software and its engineering → Software performance**; **Software verification and validation**; **System modeling languages**.

## KEYWORDS

Software Performance, Model-Driven Engineering, Performance Testing, Benchmarking.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652884

**ACM Reference Format:**
Vittorio Cortellessa. 2024. 25+ years of Software Performance: From Integrated System Modelling to ML-based Analysis, What's Next?. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3629527.3652884

## SHORT BIO

Vittorio Cortellessa is Professor at the Department of Computer Science and Engineering, and Mathematics of University of L'Aquila. He had received his Ph.D. in Computer Science at University of Roma Tor Vergata in 1995. Between 1996 and 1999 he held postdoc positions at the same institution and at European Space Agency. In 2000 and 2001 he has been Research Assistant Professor at the Computer Science and Electrical Engineering Department of West Virginia University. Since 2022 he is at University of L'Aquila. His main research interests are in the areas of Software Performance, Software Reliability, and Model-Driven Engineering. He has published more than 120 papers on international conferences and journals in these areas, and he has co-authored a monographic book on Software Performance. He has served and serves in program committees and editorial boards of conference and journals in the Software Engineering domain. He is currently Co-Chair of the Steering Committee of ACM/SPEC International Conference on Performance Engineering (ICPE) and member of the Steering Committee of IEEE International Conference on Software Architecture (ICSA). More information at: http://people.disim.univaq.it/cortelle/.

## ACKNOWLEDGEMENTS

This work is fully supported by Italian Government (Ministero dell'Università e della Ricerca, PRIN 2022 PNRR): "RECHARGE: monitoRing, tEsting, and CHaracterization of performAnce ReGrEssions" (cod.P2022SELA7).

# HetSim: A Simulator for Task-based Scheduling on Heterogeneous Hardware

Marcel Lütke Dreimann
Universität Osnabrück
Osnabrück, Germany
marcel.luetkedreimann@uos.de

Birte Friesel
Universität Osnabrück
Osnabrück, Germany
birte.friesel@uos.de

Olaf Spinczyk
Universität Osnabrück
Osnabrück, Germany
olaf@uos.de

## ABSTRACT

Server hardware is becoming more and more heterogeneous, with an increasingly diverse landscape of accelerators such as GPUs, FPGAs, or novel processing-in-memory (PIM) technologies. Designing and evaluating scheduling algorithms for these is far from trivial due to accelerator-specific setup costs, compute capabilities, and other characteristics. In fact, many existing scheduling simulators only consider some of these characteristics, or only support a specific sub-set of accelerators. To overcome these challenges, we present *HetSim*, a modular simulator for task-based scheduling on heterogeneous hardware. HetSim enables research on online and offline scheduling and placement strategies for modern compute platforms that combine CPU cores with multiple GPU, FPGA, and PIM accelerators. It is efficient, fair, and compatible with a variety of common workload descriptions, output metrics, and visualization tools. We use HetSim to reproduce results from Alebrahim and Ahmad, and examine how accelerator characteristics affect the performance of various scheduling strategies. Our results indicate that ignoring accelerator characteristics during simulation is often detrimental, and that the ideal scheduling algorithm for a given workload may depend on available accelerators and their characteristics. HetSim is available as open-source software.

## CCS CONCEPTS

• **General and reference** → **Evaluation**; **Experimentation**; • **Software and its engineering** → *Scheduling*; *Memory management*; • **Human-centered computing** → Visual analytics; • **Computing methodologies** → Discrete-event simulation.

## KEYWORDS

Simulator, Heterogeneous Hardware, Scheduling

**ACM Reference Format:**
Marcel Lütke Dreimann, Birte Friesel, and Olaf Spinczyk. 2024. HetSim: A Simulator for Task-based Scheduling on Heterogeneous Hardware. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3652275

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652275

## 1 INTRODUCTION

Server hardware has become more and more heterogeneous over recent years. In addition to many-core processors, it can now also include multiple GPUs or FPGAs. Moreover, new processing-in-memory technologies such as UPMEM PIM are emerging and becoming available for end users. Each of these accelerators has its own characteristics and limitations: it can achieve outstanding performance on workloads that it was designed for while being of limited use in other cases.

One common limitation is the need to allocate a CPU core for setup purposes. Consider the trace of PolyBench's *2mm* OpenCL benchmark [11] shown in Fig. 1. Even though the benchmark target is a GPU, OpenCL first spends 200 ms with CPU-only setup functions. The workload itself (*mm2_kernel2*) makes up less than 25 % of execution time [1] and is the only component that actually uses the GPU.

Hence, setup cost is far from negligible when designing or evaluating scheduling algorithms. This also applies to other accelerators. For instance, dynamic offloading with FPGAs relies on costly dynamic reconfiguration, and executing tasks on UPMEM PIM is also far from instantaneous [9].

Similarly, accelerators can use dedicated memory or share it with CPU cores. The former may require data transfers between main memory and accelerator memory. This heterogeneity and diversity of execution components poses new challenges for scheduling and placement decisions. Depending on the software layer where they are implemented, these can affect developers of operating systems, database management systems, language runtimes, or even applications.

Currently, integrating accelerators into applications is usually up to application developers. Novel system software like MxTasking [16] and userspace frameworks like StarPU [4] aim to help them with this task. These projects rely on *tasks* as control flow abstractions and offer a unified framework for programming different kinds of accelerators. They also deal with scheduling decisions that come up if tasks can be executed on several different accelerators. The increasing complexity and heterogeneity of accelerators leads to increasing complexity in dealing with these decisions.

The contributions of this paper are three-fold.

- We provide a discussion of challenges that designers and performance evaluation methods of scheduling strategies for heterogeneous hardware face (Section 3).
- We present *HetSim*, a simulator for task execution on heterogeneous hardware (Section 4). It is capable of evaluating

[1] While *clBuildProgram* latency can be reduced by using pre-compiled GPU binaries in this particular example, the discrepancy between setup cost (CPU) and execution time (GPU) remains.

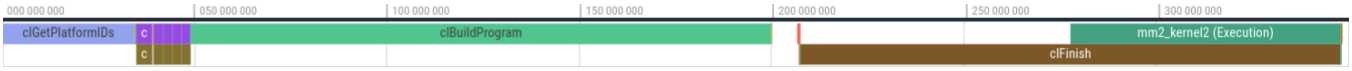



**Figure 1: Execution trace of PolyBench's *2mm* OpenCL benchmark [11] on an integrated Intel UHD Graphics Xe GPU. The benchmark uses 16 execution units; the trace has been extracted by the OpenCL Intercept Layer [3].**

scheduling algorithms on user-defined hardware platforms without the need for real-world measurements. In addition to a set of built-in workloads, it also accepts workload definitions provided by users or by existing DAG workload generators. An optional event log containing task phases and memory transfers allows for user-friendly visualization with existing tools such as *Perfetto*. With HetSim, we are able to reproduce results from Alebrahim and Ahmad.

- We use HetSim to examine how accelerator-specific overhead affects the performance of different scheduling strategies (Section 5).

The next section examines related work and outlines the gaps that HetSim aims to fill. We then cover our three contributions (see above) and conclude in Section 6.

## 2 RELATED WORK

The literature offers a variety of scheduling simulators for a diverse set of use cases.

Suranauwarat presents a simulator that exclusively deals with single-CPU scheduling algorithms [20]. It is meant to be used for educational purposes and comes with graphical animations, but does not support heterogeneous hardware.

*Realtss* focuses on evaluating real-time CPU scheduling policies without having to implement them in a real-time operating system [8]. It has education and research in mind.

*ScSF* is intended for simulation-based high-performance computing (HPC) scheduling research [18]. It offers tools for workload modeling and generation, system simulation, comparative workload analysis, and experiment orchestration.

*SimGrid* simulates distributed applications in grid environments [7]. It focuses on communication rather than computation, with latency- and bandwidth-bound links between nodes and bandwidth sharing in case of simultaneous transfers. While it can handle heterogeneous hardware, its *host resource* does not support accelerator-specific behavior. Moreover, it measures performance based on floating-point operations per second (FLOPS). We will explain the drawback of this metric in Section 3.5.

Due to SimGrids's detailed communication simulation and ease of use, several simulators build on top of its API. For instance, *Alea* focuses on event-based scheduling of heterogeneous jobs on heterogeneous resources with dynamic runtime changes [14]. It gathers information about resource status and simulation results, which can be visualized later.

Even though many simulators can gather statistics and offer visualizations, none that we are aware of support *heterogeneous* hardware with *accelerator-specific* behavior. They do not consider setup or reconfiguration costs, and only provide limited statistics and simulator output. Moreover, many simulators have been tailor-made for specific classes of scheduling strategies or hardware components, limiting their re-usability. Our contribution, *HetSim*, fills these gaps by dealing with accelerator-specific attributes in arbitrary schedules and hardware components. Combined with its support for third-party workload generation and visualization tools, this makes it more flexible than existing simulators.

## 3 DESIGN AND EVALUATION CHALLENGES

As mentioned in the introduction, modern frameworks for the management of heterogeneous computing resources use a control flow abstraction called *tasks*: self-contained units of work that cannot be preempted. The advantage of tasks over traditional abstractions such as POSIX *threads* or GPU/PIM *kernels* is their simplicity, which allows for having a unified control flow model for all supported accelerators.

While this addresses one common issue when dealing with heterogeneous accelerators, task-based scheduling algorithms still face a variety of challenges. We will now discuss the five most prominent ones that we have identified: memory heterogeneity, accelerator heterogeneity, task setup, driver frameworks, and the hardware model.

### 3.1 Memory Heterogeneity

While most accelerators come with dedicated memory, they can also share memory and even the last-level cache with the CPU – integrated GPUs such as Intel's UHD 630 are a prominent example for this. Additionally, only some accelerators with dedicated memory support direct memory access (DMA). So, a task will incur different amounts of data transfer overhead depending on where it is scheduled. At the same time, in both cases, subsequent tasks scheduled on the same accelerator may benefit from already-present data or warm caches. So, the performance of an individual task depends not just on its own schedule, but also on the schedule of preceding tasks that access shared data. Grouping those onto accelerators with a shared cache can improve performance or save energy [10].

### 3.2 Accelerator Heterogeneity

Accelerators are not one-size-fits-all devices: GPUs excel at parallel tasks that access shared memory, whereas UPMEM PIM works best with embarrassingly parallel workloads without synchronization or shared memory [17]. Executing a task on an unsuitable accelerator may result in worse performance than just running it on a CPU core [9]. Scheduling strategies have to take this into account, and simulators must be aware of it as well in order to provide useful results. This is a balance act between general-purpose algorithms and simulators that assume anything is supported anywhere, and the more complex task of encoding and utilizing knowledge about the specific properties of each accelerator.

### 3.3 Task Setup

Many accelerators rely on CPU support just like conventional peripheral devices. A CPU core has to allocate (part of) the accelerator, transfer the task's program code and possibly data, start the task, and handle communication (e.g. waiting for the task to finish and retrieving results). Both scheduling strategies and simulators must take this reliance on CPU cores into account, especially for short tasks where the setup time may exceed actual task execution (cf. Fig. 1). This is another balance act: depending on a task's attributes, using a less suitable accelerator or plain CPU execution may still be faster than the incurred CPU-bound setup cost.

### 3.4 Driver Frameworks

All accelerators that we are aware of rely on an accelerator-specific driver framework in order to execute tasks. Vendors recommend their own software for optimal performance, e.g. CUDA[2], OneAPI[3], or the UPMEM SDK[4]. However, when updates are applied to this software, the accelerator characteristic may change. Driver optimizations can reduce execution time on the accelerator, or affect the setup time of a task. If scheduling algorithms are to be evaluated on real hardware within a scheduling framework, the scheduling framework must implement the interfaces to all possible drivers. While OpenCL comes close by providing a standard (including a programming language) that encompasses a variety of accelerator types, some features and novel technologies such as UPMEM PIM are missing.

### 3.5 Hardware Model

Scheduling algorithms and simulators rely on a hardware model to determine the suitability of a given accelerator for a given task. While metrics such as clock frequency or FLOPS may work well in homogeneous settings (e.g. scheduling tasks on a CPU with slow efficiency and fast performance cores), they are insufficient for our purpose. Different accelerators may use different architectures, and thus respond differently to heavy use of branch instructions, vector operations, synchronization, shared memory accesses, and similar. For instance, GPUs tend to work well with floating point math but suffer from branching-induced performance penalties, whereas UPMEM PIM excels at integer vector operations [12]. So, actual accelerator performance is a function of the type of task it executes, and not just its instruction count or a related numeric metric. FLOPS and frequency, as used by e.g. SimGrid, capture none of these nuances.

When dealing with dedicated memory, data transfer overhead must be considered as well. The interconnect used for data transfer typically has a well-defined latency and throughput. However, if multiple data transfers are executed on a single interconnect at the same time, the bandwidth is shared.

## 4 HETSIM OVERVIEW

We now present our main contribution: *HetSim*, an event-based, discrete scheduling simulator that addresses the aforementioned challenges. It follows a modular design to allow for easy adjustments and extensions. Fig. 2 shows its main components and how they interact with each other.

The `Accelerator` base class implements common functionality and checks, such as per-task memory allocations or determining whether a given task actually has an implementation for this accelerator. Its derived classes implement accelerator-specific behavior. Each `Accelerator` instance references a single `MemoryPool` that represents the accelerator's memory and implements data object allocation and movement. Multiple accelerators can share the same memory pool to represent shared memory environments.

The `Environment` class stores a list of all accelerators (populated by `ModelLoader`) and workload tasks (generated by `TaskSetGenerator`). Each `Task` has a list of dependencies and accessed data objects; a single data object can be referenced by multiple tasks. Lastly, the simulator provides a `Profiler` that keeps track of all task phases and memory transfers. The resulting statistics can be printed on the console or saved in Google's trace format[5].

The following subsections describe how HetSim supports the analysis of scheduling and placement strategies for heterogeneous systems. We will cover workload generation, the simulation process itself, and simulation output.

### 4.1 Workload Generation

When evaluating scheduling algorithms, it is important to ensure that workloads resemble a wide range of real-world applications and contain randomized components to identify systematic errors in scheduling strategies. At the same time, it is desirable to obtain deterministic simulation results for debugging and reproduction purposes.

HetSim achieves both by providing a set of random number generators that can alter workload attributes, and storing the random seeds and other workload parameters of each simulation run in a `Config` object (cf. Fig. 2). Randomizable attributes include the amount and size of data objects accessed by a task, its expected runtime, and its set of supported accelerators. Each simulation run can be reproduced by loading the associated hardware model description and configuration into a subsequent HetSim invocation.

On top of this, HetSim supports four different types of task sets. It can also simulate mixed workloads that combine multiple task set types.

**Random** task sets make use of all possible characteristics an application can have, including data (de)allocation and dependencies between tasks. The latter can be disabled if the increased scheduling complexity caused by task dependencies is undesirable.

**Database Queries** often serve as motivation for heterogeneous hardware, as database operators can benefit from significant performance boosts by utilizing GPU, FPGA or PIM accelerators [5, 15, 19]. The task set is composed of query and aggregation tasks. Each query task accesses the same large data object (the database) and allocates a smaller data object for its result, and the aggregation tasks combine those intermediate results into a single data object.

**Directed Acyclic Graphs** (DAGs) are often used for evaluating scheduling strategies such as *Heterogeneous Earliest Finish Time* (HEFT) [2, 13]. HetSim supports the DAG task set format of two

[2]https://developer.nvidia.com/cuda-toolkit
[3]https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html
[4]https://sdk.upmem.com/
[5]https://docs.google.com/document/d/1CvAClvFfyA5R-PhYUmn5OOQtYMH4h6I0nSsKchNAySU

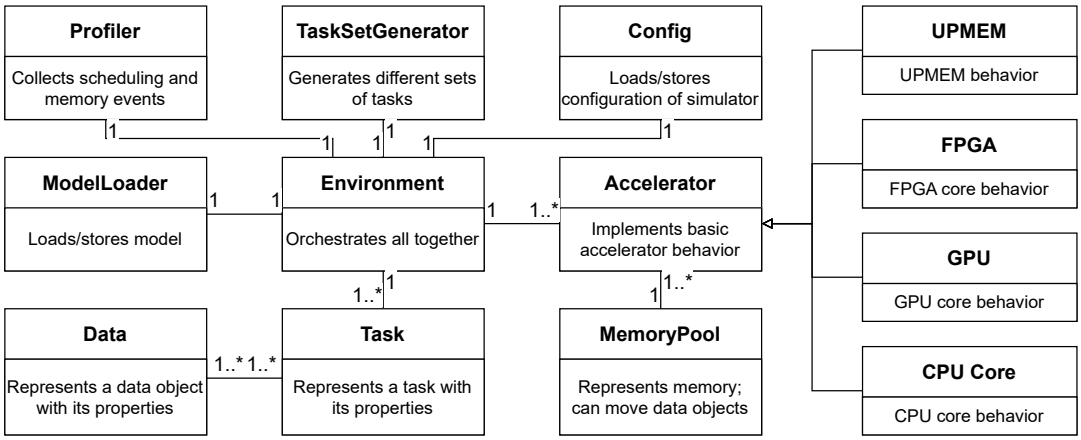



**Figure 2: Overview over the most important simulator components.**

existing tools: DAGGEN and Pegasus. DAGGEN generates synthetic, random task graphs that are intended for CPU scheduling [21]. Pegasus workloads, on the other hand, are based on real scientific workloads like RNA analysis or the characterization of earthquake hazards [6]. In both cases, each graph node represents a task, and each edge encodes a dependency between two tasks. DAGGEN associates tasks with random computation costs, whereas Pegasus workloads do not provide any runtime annotation. Hence, HetSim generates new computation costs for accelerators according to the provided simulator configuration, and annotates graph edges with communication costs.

**User-defined** task sets allow for evaluating scheduling strategies that are optimized for specific workloads, and for analyzing the performance and acceleration potential of applications rather than scheduling algorithms. Users can simulate applications that do not yet make use of heterogeneous hardware with a fixed scheduling strategy, and see how enabling the execution of tasks on specific accelerators changes application runtime and resource utilization.

## 4.2 Simulation

HetSim is implemented as a C++ library. This way, developers can easily port existing scheduling algorithms to the simulator, or implement new algorithms that can later be used in real systems. Additionally, it allows for the use of debugging tools to analyze scheduling algorithms in detail. HetSim supports both online and offline scheduling algorithms. It is available as open source software and extensively documented, thus enabling developers to extend its set of accelerators with custom implementations.

In addition to scheduling algorithm (C++) and task set (Sec. 4.1), users must specify the simulated hardware platform. Such a platform description consists of four elements: memory pools, accelerator architectures, the accelerators themselves, and links between memory pools. HetSim uses an XML format for platform descriptions in order to remain flexible when it comes to supporting future heterogeneous hardware platforms. Listing 1 shows an example; we will now describe its four components in detail.

A **MemoryPool** represents main memory (DRAM), a specific accelerator's dedicated memory, or similar. The memory model is not limited to volatile RAM: it also supports persistent memory such as Optane DCPMMs and SSDs. Each memory pool must have a unique ID and a maximum size.

```
<MemoryPool id="0" size="16384" />
<MemoryPool id="1" size="8192" />
<Architecture id="0" perf_min="1.0" perf_max="1.5" />
<Architecture id="1" perf_min="1.8" perf_max="2.0" />
<Accelerator id="0" archid="0" memorypool="0" type="CPUCore" setupcost="0"
        dma="1" modelerror="0.0" poweridle="10.0" powerload="25.0" />
<Accelerator id="1" archid="0" memorypool="0" type="CPUCore" setupcost="0"
        dma="1" modelerror="0.0" poweridle="10.0" powerload="25.0" />
<Accelerator id="2" archid="1" memorypool="1" type="GPU" setupcost="250" dma=
        "0" modelerror="0.0" poweridle="20.0" powerload="100.0" />
<Link src="0" dst="1" speed="100" />
<Link src="1" dst="0" speed="100" />
```

**Listing 1: Excerpt of a hardware platform description.**

An **Architecture** encodes performance attributes of a specific accelerator architecture, e.g. a specific GPU model. It has a unique ID as well as minimum and maximum performance factors. HetSim uses these to generate randomized computation costs based on the interval defined by the abstract performance factors. Overlapping performance ranges of architectures allow for non-linear computation costs. For example, consider two architecture with performance ranges $[1.0, 2.0]$ and $[1.5, 2.5]$. Tasks will most likely run faster on the second architecture due to its larger performance factors. However, because of the overlap $[1.5, 2.0]$, tasks can also be equally fast or the first architecture can be faster than the second. This way, HetSim can break out of simple FLOPS-based performance models like the one used in SimGrid.

An **Accelerator** is a concrete compute unit instance, e.g. a CPU core or an FPGA. It references an accelerator architecture and a memory pool by ID, and also has its own ID. The `type` indicates whether it is a CPU, GPU, FPGA, UPMEM, or other kind of accelerator. Additional parameters indicate the absolute setup cost for a task (recall Fig. 1), DMA support, model error, and power usage in idle as well as under load.

The idea behind model error is that the expected execution times of individual tasks may be inaccurate. To examine how susceptible a scheduling algorithm is to such inaccuracies, users can set

`modelerror` to a value within $[0.0, 1.0)$ that controls the dispersion of actual task execution time around the expected execution time. For instance, with `modelerror="0.1"`, actual execution time can be up to 10 % lower or higher than expected execution time.

The `poweridle` and `powerload` values are used by HetSim to estimate the power consumption of each individual accelerator and of the entire system. In order to define virtual accelerators, the corresponding accelerator can be duplicated with the same `memorypool` ID but a new accelerator ID.

Finally, each **Link** encodes a link between two memory pools and its data transfer speed. Duplex communication is modeled by two Link elements with individual bandwidths.

### 4.3 Statistics and Visualization

Adequate evaluation metrics and statistics as well as intuitive visualization are crucial for understanding and analyzing scheduling algorithms and application performance. HetSim supports a variety of those.

*Metrics.* Given a set of accelerators $A$ and a schedule *plan*, HetSim provides its simulated total execution time (*makespan*) as well as the *SLR*, *Speedup* and *Efficiency* metrics from the literature [1].

SLR (Schedule Length Ratio) is often used to compare schedules in a way that is independent of DAG topology. It divides the makespan by the sum of task execution times on the critical path. Hence, a better strategy has a lower SLR.

$$SLR(plan) = \frac{makespan(plan)}{\sum_{ti \in CP} \min_{a \in A} w_{ti,a}}$$

Speedup describes how much the schedule benefits from using multiple accelerators. It divides the fastest sequential execution time that can be achieved when scheduling all tasks on a single accelerator by the makespan.

$$Speedup(plan) = \frac{\min_{a \in A} \sum_{ti \in T} w_{ti,a}}{makespan(plan)}$$

Efficiency, in turn, describes how well the scheduling algorithm utilizes the accelerators. It is defined as the ratio of speedup over the number of available accelerators $|A|$. An efficiency of 1.0 indicates that the total execution time of a task set is evenly split across all available accelerators.

$$Efficiency(plan) = \frac{Speedup(plan)}{|A|}$$

*Task Phases and Visualization.* HetSim records all memory transfers and, for each task, which of the following phases it is currently in:

- Task setup
- Task blocked (waiting for dependency)
- Task blocked (waiting for setup)
- Setup blocked (accelerator busy)
- Data transfer
- Task execution

For each phase, it records start time, duration, and affected accelerator. For memory transfers, it also records the utilization of the respective memory pool.

This allows HetSim to determine the accumulated time that each accelerator has spent in the different task phases as well as its total number of tasks and load. The recorded phases can be visualized with Google's *Perfetto* as shown in Fig. 3. Perfetto generates an interactive plot similar to a Gantt diagram and provides utilities for detailed analysis.

HetSim also determines the minimum, maximum, and average wall-clock time that the machine running the simulation spent making scheduling decisions. Simulation overhead is excluded from this *decision time*. Thus, users can compare the overhead of different scheduling algorithms.

## 5 EXAMPLE AND EVALUATION

We will now analyze four scheduling algorithms (three online, one offline) to demonstrate the scientific usefulness, performance, and correctness of HetSim. Those are heterogeneous Round Robin (hRR), GreedyET, GreedyDS, and HEFT. Source code, data, and analysis scripts are available at https://ess.cs.uos.de/git/artifacts/wosp-c-2024-hetsim-artifacts.

hRR is a naïve adaptation of the Round Robin (RR) CPU scheduling algorithm for heterogeneous hardware. It does not consider performance attributes, and simply iterates over all accelerators until it has found one that is capable of executing the task. GreedyET picks the accelerator with the shortest expected execution time, and GreedyDS additionally takes data transfers and setup phases into account. HEFT is an offline scheduling algorithm from the literature [2, 13]. All algorithms use CPU core 0 to run setup code for task execution on a GPU or FPGA, if needed. An important distinction is the ability to balance load across CPU cores and accelerators. In contrast to hRR and HEFT, GreedyET and GreedyDS are not able to do load balancing and might over-utilize preferred accelerators.

### 5.1 Performance of scheduling strategies

Our evaluation uses the Sipht[6] Pegasus task set. It consists of 33 tasks with dependencies from the genome analysis / RNA translation domain. The hardware platform has eight CPU cores in two different NUMA regions of 64 GiB each, an integrated GPU sharing the memory with node 0, and a dedicated GPU and FPGA with 16 GiB of memory each. The link between the NUMA regions has a bandwidth of 100 MB/s and all other links can transfer data with 25 MB/s.

We examine four configurations: a complex model with accelerator-specific setup times, and a simple model without those. Both come in two flavors: slow (little speedup provided by accelerators) and fast (up to two times higher speedup). Table 1 lists the performance intervals and setup costs, and Table 2 shows a HetSim configuration excerpt for the complex models. In the simple cases, `min_fpga_reconf_time` and `max_fpga_reconf_time` are set to 0. We use the SLR metric from the literature to compare the strategies. Note that this metric does not take setup costs into account when calculating the critical path. An adjusted SLR definition for heterogeneous hardware might be useful in the future.

On the simple model without setup costs, HEFT performs best (see Fig. 4). As an offline algorithm, it has prior knowledge of all tasks, so this is to be expected. Moreover, HEFT's internal model is close to our simple model, with the only exception being that HEFT allows concurrency between memory transfers and task execution.

[6] https://pegasus.isi.edu/workflow_gallery/gallery/sipht/index.php

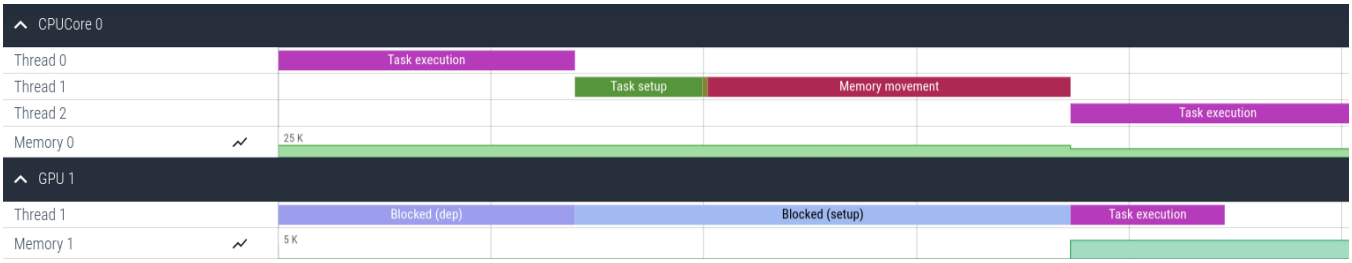



**Figure 3: Small example simulator trace visualized by Perfetto.**

| Accelerator | slow | fast | setup time |
|---|---|---|---|
| CPU Core | [1.0, 1.0] | [1.0, 1.0] | 0 |
| iGPU | [0.8, 2.2] | [0.8, 4.4] | 100 |
| dGPU | [1.1, 3.6] | [1.1, 7.2] | 120 |
| FPGA | [1.2, 4.0] | [1.2, 8.0] | 90 |

**Table 1: Performance intervals [perf_min, perf_max] and setup times for "slow" and "fast" configurations.**

| Setting | | Setting | |
|---|---|---|---|
| max_fpga_reconf_time | 200 | strict_exceptions | 1 |
| min_fpga_reconf_time | 100 | min_data_size | 4 |
| min_task_runtime | 10 | max_data_size | 300 |
| max_task_runtime | 500 | max_task_delay | 0 |
| partial_task_families | 0 | seed_uniform | 3 |
| min_data_objects | 1 | seed_task_runtime | 1 |
| max_data_objects | 5 | seed_data_size | 2 |

**Table 2: A simulator configuration that allows for FPGA reconfiguration. The seeds are different for each iteration.**

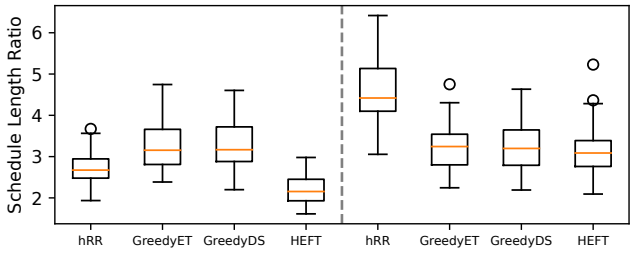



**Figure 4: SLR for scheduling strategies without setup phases, using slow (left) and fast (right) accelerators.**

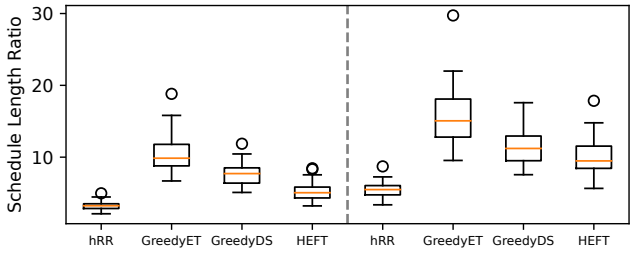



**Figure 5: SLR for scheduling strategies with setup phases, using slow (left) and fast (right) accelerators.**

In the slow flavor (left), hRR comes in second, and the greedy strategies perform worst. HetSim output and Perfetto diagrams reveal that hRR benefits from load balancing, whereas the greedy strategies execute all tasks on dGPU and FPGA and leave the (slower) CPU and iGPU idle. In the fast flavor (right), on the other hand, the greedy strategies outperform hRR. Here, hRR leaves accelerators idle, as its round-robin algorithm treats eight (slow) CPU cores and three (fast) accelerators as eleven equally fast compute nodes.

Fig. 5 shows the impact of including setup costs in the models. Now, hRR is best in both flavors, outperforming even the offline HEFT approach. hRR prefers CPU cores (see above) and thus incurs little CPU/FPGA setup costs, whereas HEFT chooses GPU or FPGA execution without taking setup cost into account, degrading its performance. Additionally, for HEFT and both greedy algorithms, CPU core 0 is overloaded with setup code for GPU and FPGA tasks, thus delaying their start. While GreedyET favors the FPGA due to having the lowest execution time for most tasks, it still comes up with a suboptimal schedule due to its high reconfiguration time. GreedyDS performs better, but is still behind hRR and HEFT due to its lack of load balancing and over-utilization of CPU core 0. Overall, we see that mapping tasks to devices is far from trivial. Any performance gains provided by an accelerator can easily be nullified by its setup costs.

Another interesting insight is that the overall SLR of every algorithm has increased in the fast flavor. This does not mean that the makespan has increased, but rather that the length of the critical path across the DAG was reduced. All of these findings could be highly useful during research and development of scheduling and placement strategies.

## 5.2 Costs of scheduling decisions

Of course, more complex scheduling decisions incur higher scheduling overhead. Fig. 6 shows the time per scheduling decision on the Sipht task set, measured on the simulating machine. hRR, with its lack of model usage, is fastest. GreedyET is five times slower, and GreedyDS's fine-grained approach results in an up to 20-fold increase in overhead.

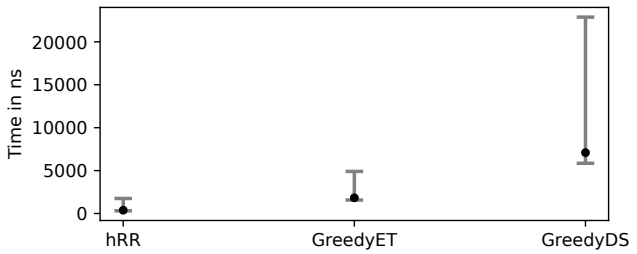



**Figure 6: Benchmark of scheduling algorithms with minimum, maximum and average decision time measured on the simulating machine (i7-11850H).**

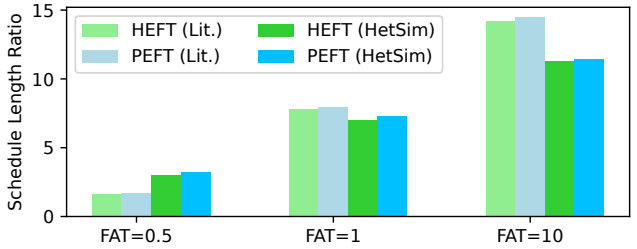



**Figure 7: SLR results for HEFT and PEFT by Alebrahim and Ahmad (annotated as "Lit.") and Hetsim.**

## 5.3 Simulation Correctness

To evaluate the correctness of HetSim, we used it to reproduce HEFT and PEFT SLRs reported by Alebrahim and Ahmad. Those build upon a hardware model with two processing units with different performance values for each task and without setup cost. The input task sets were generated with DAGGEN with variable levels of parallelism ("FAT" parameter). The HetSim reproduction uses a hardware model consisting of two CPU cores with different architectures and identical DAGGEN parameters. As Fig. 7 shows, while absolute values differ slightly due to a different simulation model, the relation between HEFT and PEFT remains the same.

When comparing HetSim's plans with the example given by Alebrahim and Ahmad, we see only a single difference. HetSim's simulation model does not allow memory transfer during task execution, while the HEFT and PEFT plan from the paper does.

## 5.4 Simulator performance

Lastly, we evaluated the performance of HetSim itself by using the database query benchmark with the hRR algorithm. As Fig. 8 shows, simulation time scales linearly with the number of tasks, with a maximum of just 2.2 *s* for 500,000 tasks. Despite its single-threaded implementation, HetSim is fast enough to simulate large task sets in a reasonable amount of time. In practice, the simulation time was not a limitation for us, as several simulations can run on different cores at the same time.

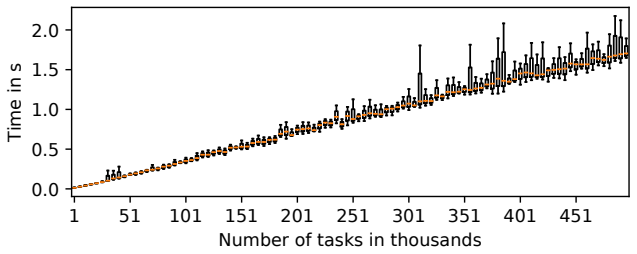



**Figure 8: Simulation time for large task sets on an i7-11850H.**

## 6 CONCLUSION AND FUTURE WORK

We have presented HetSim, a scheduling simulator for heterogeneous hardware that allows developers to study and analyze scheduling algorithms for given workloads and hardware configurations. We showed that it can help identify weaknesses of scheduling algorithms. The simulator records statistics and allows visualization of schedules by third-party tools. Furthermore, HetSim supports benchmarking scheduling decisions to help developers evaluate the performance overhead of complex algorithms. The simulator handles large simulations with hundreds of thousands of tasks in only a few seconds and can reproduce results from the literature. HetSim is freely available at https://ess.cs.uos.de/git/software/hetsim under an open source license. It can serve as a reusable evaluation tool that removes the need for writing algorithm-specific simulators.

The development of HetSim is ongoing. We plan to handle contention and bandwidth limitations if multiple data transfers happen at the same time on a single link in our simulator. HetSim could also use a real hardware model without abstract performance values to estimate absolute execution times on real hardware instead of time units. Furthermore, we plan to extend the possible use cases of HetSim. For example, our simulator could also be used to automatically explore the design space of scheduling strategies for a given hardware model.

## ACKNOWLEDGMENTS

The work on this paper has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 361498541, 502565817.

## REFERENCES

[1] Wakar Ahmad, Bashir Alam, and Sahil Malik. 2019. Performance analysis of list scheduling algorithms by random synthetic DAGs. In *Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE)*. https://doi.org/10.2139/ssrn.3349016

[2] Shaikhah Alebrahim and Imtiaz Ahmad. 2017. Task Scheduling for Heterogeneous Computing Systems. *J. Supercomput.* 73, 6 (jun 2017), 2313–2338. https://doi.org/10.1007/s11227-016-1917-2

[3] Ben Ashbaugh. 2018. Debugging and Analyzing Programs Using the Intercept Layer for OpenCL Applications. In *Proceedings of the International Workshop on OpenCL* (Oxford, United Kingdom) *(IWOCL '18)*. Association for Computing Machinery, New York, NY, USA, Article 14, 2 pages. https://doi.org/10.1145/3204919.3204933

[4] Cédric Augonnet, Olivier Aumage, Nathalie Furmento, Raymond Namyst, and Samuel Thibault. 2012. StarPU-MPI: Task Programming over Clusters of Machines Enhanced with Accelerators. In *Recent Advances in the Message Passing Interface*, Jesper Larsson Träff, Siegfried Benkner, and Jack J. Dongarra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 298–299.

[5] Alexander Baumstark, Muhammad Attahir Jibril, and Kai-Uwe Sattler. 2023. Processing-in-Memory for Databases: Query Processing and Data Transfer. In

*Proceedings of the 19th International Workshop on Data Management on New Hardware* (Seattle, WA, USA) *(DaMoN '23)*. Association for Computing Machinery, New York, NY, USA, 107–111. https://doi.org/10.1145/3592980.3595323

[6] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. 2008. Characterization of scientific workflows. In *2008 Third Workshop on Workflows in Support of Large-Scale Science*. 1–10. https://doi.org/10.1109/WORKS.2008.4723958

[7] H. Casanova. 2001. Simgrid: a toolkit for the simulation of application scheduling. In *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. 430–437. https://doi.org/10.1109/CCGRID.2001.923223

[8] Arnoldo Diaz, Ruben Batista, and Oskardie Castro. 2007. Realtss: a real-time scheduling simulator. In *2007 4th International Conference on Electrical and Electronics Engineering*. 165–168. https://doi.org/10.1109/ICEEE.2007.4344998

[9] Birte Friesel, Marcel Lütke Dreimann, and Olaf Spinczyk. 2023. A Full-System Perspective on UPMEM Performance. In *Proceedings of the 1st Workshop on Disruptive Memory Systems* (Koblenz, Germany) *(DIMES '23)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3609308.3625266

[10] Vıctor Garcıa, Juan Gomez-Luna, Thomas Grass, Alejandro Rico, Eduard Ayguade, and Antonio J. Pena. 2016. Evaluating the effect of last-level cache sharing on integrated GPU-CPU systems with heterogeneous applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. 1–10. https://doi.org/10.1109/IISWC.2016.7581277

[11] Scott Grauer-Gray, Lifan Xu, Robert Searles, Sudhee Ayalasomayajula, and John Cavazos. 2012. Auto-tuning a high-level language targeted to GPU codes. In *2012 innovative parallel computing (InPar)*. Ieee, 1–10.

[12] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. 2022. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access* 10 (2022), 52565–52608. https://doi.org/10.1109/ACCESS.2022.3174101

[13] Julien Herrmann, Loris Marchal, and Yves Robert. 2014. Memory-Aware List Scheduling for Hybrid Platforms. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. 689–698. https://doi.org/10.1109/IPDPSW.2014.80

[14] Dalibor Klusâcek and Hana Rudovâ. 2010. Alea 2: job scheduling simulator. ICST. https://doi.org/10.4108/ICST.SIMUTOOLS2010.8722

[15] Mehdi Moghaddamfar, Christian Färber, Wolfgang Lehner, Norman May, and Akash Kumar. 2021. Resource-Efficient Database Query Processing on FPGAs. In *Proceedings of the 17th International Workshop on Data Management on New Hardware* (Virtual Event, China) *(DAMON '21)*. Association for Computing Machinery, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3465998.3466006

[16] Michael Müller, Thomas Leich, Thilo Pionteck, Gunter Saake, Jens Teubner, and Olaf Spinczyk. 2020. He..ro DB: A Concept for Parallel Data Processing on Heterogeneous Hardware. In *Architecture of Computing Systems – ARCS 2020*, André Brinkmann, Wolfgang Karl, Stefan Lankes, Sven Tomforde, Thilo Pionteck, and Carsten Trinitis (Eds.). Springer International Publishing, Cham, 82–96.

[17] Joel Nider, Craig Mustard, Andrada Zoltan, John Ramsden, Larry Liu, Jacob Grossbard, Mohammad Dashti, Romaric Jodin, Alexandre Ghiti, Jordi Chauzi, and Alexandra Fedorova. 2021. A Case Study of Processing-in-Memory in off-the-Shelf Systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 117–130. https://www.usenix.org/conference/atc21/presentation/nider

[18] Gonzalo P. Rodrigo, Erik Elmroth, Per-Olov Östberg, and Lavanya Ramakrishnan. 2018. ScSF: A Scheduling Simulation Framework. In *Job Scheduling Strategies for Parallel Processing*, Dalibor Klusáček, Walfredo Cirne, and Narayan Desai (Eds.). Springer International Publishing, Cham, 152–173.

[19] Viktor Rosenfeld, Sebastian Breß, and Volker Markl. 2022. Query Processing on Heterogeneous CPU/GPU Systems. *ACM Comput. Surv.* 55, 1, Article 11 (jan 2022), 38 pages. https://doi.org/10.1145/3485126

[20] Sukanya Suranauwarat. 2007. A CPU scheduling algorithm simulator. In *2007 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*. F2H–19–F2H–24. https://doi.org/10.1109/FIE.2007.4417885

[21] Frédéric Suter and Sascha Hunold. 2013. Daggen: A synthetic task graph generator.

# Privacy-Preserving Sharing of Data Analytics Runtime Metrics for Performance Modeling

Jonathan Will
will@tu-berlin.de
Technische Universität Berlin
Berlin, Germany

Dominik Scheinert
dominik.scheinert@tu-berlin.de
Technische Universität Berlin
Berlin, Germany

Seraphin Zunzer
zunzer@campus.tu-berlin.de
Technische Universität Berlin
Berlin, Germany

Jan Bode
jan.bode@campus.tu-berlin.de
Technische Universität Berlin
Berlin, Germany

Cedric Kring
c.kring@campus.tu-berlin.de
Technische Universität Berlin
Berlin, Germany

Lauritz Thamsen
lauritz.thamsen@glasgow.ac.uk
University of Glasgow
Glasgow, United Kingdom

## ABSTRACT

Performance modeling for large-scale data analytics workloads can improve the efficiency of cluster resource allocations and job scheduling. However, the performance of these workloads is influenced by numerous factors, such as job inputs and the assigned cluster resources. As a result, performance models require significant amounts of training data. This data can be obtained by exchanging runtime metrics between collaborating organizations. Yet, not all organizations may be inclined to publicly disclose such metadata.

We present a privacy-preserving approach for sharing runtime metrics based on differential privacy and data synthesis. Our evaluation on performance data from 736 Spark job executions indicates that fully anonymized training data largely maintains performance prediction accuracy, particularly when there is minimal original data available. With 30 or fewer available original data samples, the use of synthetic training data resulted only in a one percent reduction in performance model accuracy on average.

## CCS CONCEPTS

• **Computing methodologies** → *Distributed computing methodologies*; • **Information systems**; • **Security and privacy**;

## KEYWORDS

Distributed Dataflows, Resource Allocation, Performance Modeling, Data Sharing, Data Privacy

**ACM Reference Format:**
Jonathan Will, Dominik Scheinert, Seraphin Zunzer, Jan Bode, Cedric Kring, and Lauritz Thamsen. 2024. Privacy-Preserving Sharing of Data Analytics Runtime Metrics for Performance Modeling. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3629527.3652276

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652276

## 1 INTRODUCTION

Distributed dataflow systems, such as Apache Spark [16] and Apache Flink [2] enable parallel data processing on large clusters of commodity hardware by facilitating parallelization and error handling. Here, performance models, which accurately estimate a job's runtime on various cluster setups, enable efficient job scheduling and job-specific resource allocation [6, 14].

However, accurately modeling the performance of such data processing jobs is challenging. Excluding unpredictable events like hardware failures, various factors in the wider execution context influence runtime behavior. These factors include data analytics algorithm and job parameters, software versions, dataflow framework parameters, input dataset characteristics, the cluster resources provided, and possibly the interference of co-located jobs running on the same cluster [5, 15]. Many of these factors may vary between job executions. Therefore, due to the potentially high-dimensional feature space, creating comprehensive performance models necessitates access to substantial amounts of training data.

There are approaches for sharing execution-context-aware performance metrics among collaborators [12, 15]. Nevertheless, this approach to collaborative machine learning raises concerns about data privacy, particularly for private sector companies who may be reluctant to share such metadata with competitors. This applies especially to certain characteristics of their processed datasets that may disclose internal business information, such as a business's customer count.
Several methods have been proposed for achieving privacy in collaborative machine learning. These methods vary in effectiveness depending on the specific application [3, 4, 7, 9, 10, 13]. One promising method for sharing training data while maintaining privacy is differential privacy with data synthesis [9, 10].

In this paper, we introduce an automated method for privacy-preserving collaborative performance modeling for dataflow workloads. Our approach involves obfuscating performance model training data using differential privacy through data synthesis. Additionally, we assess and discuss the potential of this method to maintain performance model accuracy when using synthetic training data. We also measure the overhead associated with generating synthetic performance data. For the evaluation, we use a dataset of 736 unique Spark job executions.

## 2 RELATED WORK

This section explains performance modeling of distributed dataflow workloads and lays out privacy-preserving approaches to collaboratively training machine learning models.

### 2.1 Dataflow Job Performance Modeling

The performance of distributed dataflow jobs is influenced by numerous factors. These factors include the type and size of the allocated cluster, software versions, along with certain job parameters and dataset characteristics.

Previous works on cluster resource allocation establish performance models for different cluster configurations to learn dataflow job behavior. These models can be employed for automated scheduling or resource allocation decisions [5, 6, 11, 12, 14, 15].
*Karasu* [12] utilizes shared performance metrics to accelerate the iterative optimization of given objectives like minimizing runtime or carbon emissions.
*C3O* [15] shares models and performance metrics for a specific job in a single repository. The repository maintainers annotate all the job parameters and dataset characteristics that influence performance.

While data sharing approaches can help solve the cold-start problem of model-based performance optimization, a drawback of these approaches is that participating organizations must be willing to share metadata, including dataset characteristics, despite privacy concerns.

### 2.2 Privacy in Collaborative Machine Learning

For achieving training data privacy, Liu et al. have identified the following three general categories of approaches [8]:

*1. Aggregation.* Aggregation-based approaches for privacy in collaborative machine learning involve participants independently training models on their local data and sharing aggregated model updates, such as gradients or statistics instead of raw data. One prominent example is Federated Learning [7].

*2. Encryption.* Training with encrypted data has been demonstrated to be effective for relatively simple models, like Naive Bayes and decision trees [1]. Notable examples of encryption methods include Secure Multi-Party Computation [4] and Homomorphic Encryption [3].

*3. Obfuscation.* Various obfuscation techniques can be applied to unencrypted training data, including adding noise or generating new data while maintaining statistical properties necessary for training accurate models. Notable methods include Data Anonymization [13] and Data Synthesis [9, 10].

We use an obfuscation-based approach since the other two categories of approaches have shortcomings that limit their applicability to collaborative performance modeling of data analytics workloads. Aggregation-based methods necessitate continuous cooperation of several collaborators, which might not be feasible for rarely-used data analytics jobs.
Encryption-based approaches can share training data, but the limited model viability of those approaches limit their applicability to performance modeling of data analytics workloads.

## 3 APPROACH

This section outlines our obfuscation-based approach for privacy-preserving collaborative performance modeling.

### 3.1 Idea Overview

The main aim of our approach is to share performance model training data for data analytics workloads between collaborating organizations. In order to incentivize data sharing, it is important to anonymize meta information pertaining to a collaborator's workloads. At the same time, the data must maintain the statistical properties required for training accurate performance models, such as an accurate relation between execution context and runtime. We are presenting an automated method for collaborative performance modeling of dataflow workloads while preserving privacy, an overview of which is shown in Figure 1.

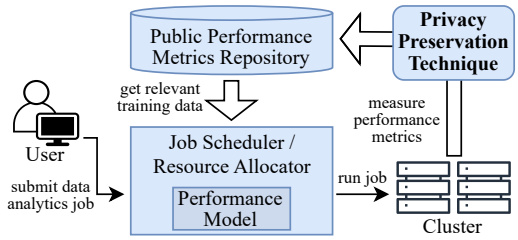



**Figure 1: High-level overview of privacy-preserving runtime metrics sharing for collaborative performance modeling.**

*Example Use Case:*
An online retailer regularly processes sales data and captures performance metrics, including runtime and execution context, such as the number of rows and columns processed and the public cloud resources used. From this data, the retailer generates synthetic datapoints that collaborators can use to train reasonably accurate performance models. Yet, collaborators cannot derive sensitive information from the shared data, such as the number of sales processed by a specific job or the total amount of sales processed during a certain time period.

### 3.2 Data Obfuscation via Data Synthesis

To facilitate privacy-preserving sharing of performance model training data, we employ an obfuscation technique introduced as DataSynthesizer by Ping et al. [9]. This technique generates synthetic data from the original dataset in two steps. Each step is represented by a separate system module.

1) The DataDescriber captures the data types, correlations, and distributions of the attributes in the original dataset and generates a data summary.

2) The DataGenerator samples any specified number of synthetic data points from the data distribution summary generated by DataDescriber. Therefore, the synthetic data points differ from the original data in terms of both quantity and content.

DataSynthesizer has been released as an open-source tool[1].

[1] `github.com/DataResponsibly/DataSynthesizer`, accessed in January 2024

# 4 EVALUATION

In this section, we assess the feasibility of our approach through experimental evaluation. We measure the accuracy of performance models trained with synthetic data and the overhead involved in producing such data.

## 4.1 Experimental Setup

In the evaluation, we use the trace dataset and the performance models published in C3O [15].

*Trace Dataset.*
The dataset contains Spark job executions for five distinct algorithms that were tested across various cluster configurations in Amazon EMR, a managed Spark service. The algorithms, namely Sort, Grep, Linear Regression, K-Means, and Page Rank, were executed on clusters of different sizes, with varying runtime-influencing job parameters and input dataset characteristics. The dataset comprises 36, 150, 140, 140, and 270 unique runtime experiments for the aforementioned jobs, respectively.

*Performance Models.*
We utilize the C3O performance modeling system's two default models: Gradient boosting and a model based on Ernest [14]. Typically, gradient boosting displays higher accuracy, except in situations with limited availability of training data, where the Ernest model may have superior accuracy.
We evaluate the trained model's accuracy with the mean absolute percentage error (MAPE) metric. For example, if the predicted runtime deviates by 20%, the MAPE will be expressed as 0.2.

*Local Hardware.*
We measured the overhead of generating synthetic data with a Python script that ran single-threaded on an octa-core AMD Ryzen 7 PRO 4750U processor (1.7-4.1 GHz, 8MB cache).

The full evaluation is available in a public code repository: `github.com/dos-group/pm-data-privacy`

## 4.2 Performance Modeling with Synthetic Data

A viable approach must allow for accurate performance modeling with synthetic data. We assess the performance model's error trained on original and synthetic data in various scenarios.

*Synthetic Training Data Size and Performance Model Accuracy.*
First, we investigate how the creation of a substantial quantity of synthetic training data impacts model accuracy. We measured the accuracy of performance models that were trained on complete original datasets for different Spark jobs. Then, we extracted various amounts of synthetic data from the same dataset, mainly larger quantities than the original data available. With this sample data, we retrained the models and measured their accuracy.

Figure 2 shows the results of this experiment. They suggest that additional synthetic training data does not have an observable impact on model accuracy beyond a certain point. Further, using synthetic data for training works differently well compared to using original data, depending on the model performance and type of job.

*Sampling Synthetic Data from Few Available Original Data Points.*
Next, we examine the feasibility of generating synthetic training data with limited availability of original training data. To this end, we randomly selected different small quantities of samples from the original performance dataset and generated 1000 synthetic data points from that. Then, we trained the performance models on both the original and synthetic data and compared their prediction accuracy.

Figure 3 shows the results of this experiment. We find that when original data availability is *low*, the use of synthetic data for training results in performance models that are nearly as accurate. For availability ranging from 3 to 30 original samples, we observed a difference of only 1.14%. However, this contrasts with the results from the previous experiment shown in Figure 2, where models trained on synthetic data sampled from a *larger* dataset of original data can perform significantly worse than models trained on the original data.

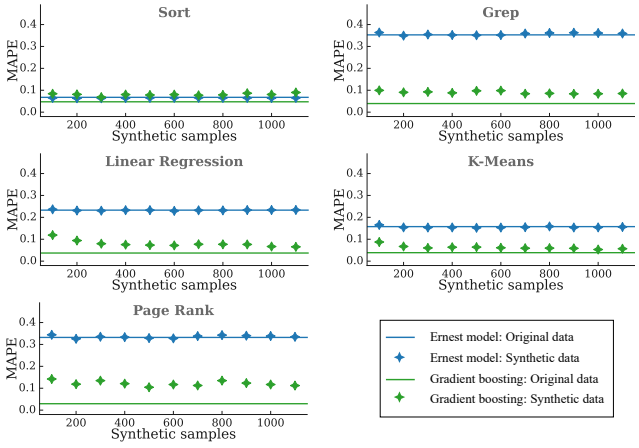



**Figure 2: Synthetic training dataset size and resulting performance model error compared to using the full original data.**

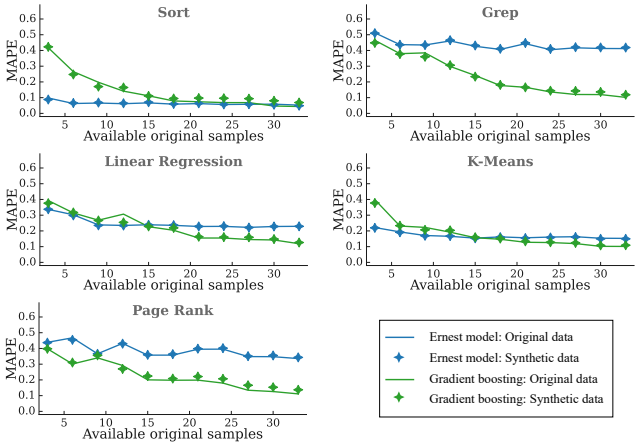



**Figure 3: Performance model error with 1000 synthetic data samples, generated from small amounts of original data.**

### 4.3 Data Synthesis Overhead

A viable approach to creating synthetic data for accurate performance modeling must not impose excessive overhead for data creation. We measured the overhead of creating synthetic training data on the hardware described in Section 4.1.

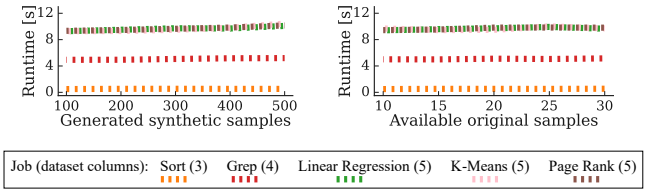



**Figure 4: Overhead for creating synthetic data for different Spark job performance datasets.**

In Figure 4, we see that for performance datasets containing the runtime and runtime-influencing factors of typical Spark jobs, this overhead was measured to be approximately between half a second and ten seconds. We observe that the computational cost of synthesizing data does not significantly increase with an increase in the amount of sampled synthetic data or the number of available samples in the original dataset. Rather, the findings suggest that the primary computational effort arises from processing each attribute, i.e., column, in the original dataset. In the case of DataSynthesizer, this part is conducted by the DataDescriber component.

### 4.4 Discussion

We will now discuss the experimental evaluation's results in terms of the practical implications for our approach's viability.

First, we observed that it is feasible to generate substantial quantities of synthetic data without compromising the model's accuracy. This implies that we can achieve privacy not just by modifying the content of each data point, but also by creating arbitrarily large amounts of synthetic data, thereby concealing the actual quantity of processed jobs.

Then, it has been observed that the model accuracy gap when using synthetic data is lowest when the quantity of original data points is low. In instances where publicly shared training data points are unavailable or rare, the introduction of synthetic data can have a significant positive effect on the model accuracy of collaborators. Consequently, sharing synthetic data is particularly advantageous in the early stages of a training data sharing initiative.

Finally, the computational overhead of generating synthetic performance data has been shown to range in seconds for performance datasets of typical Spark jobs on typical consumer hardware. This low amount of time should not discourage collaborators from generating and sharing synthetic data.

## 5 CONCLUSION

In summary, this paper has explored how differential privacy via data synthesis can facilitate the sharing of runtime data for performance modeling of data analytics workloads in a privacy-preserving manner. Our initial method has demonstrated an acceptable trade-off between model prediction accuracy and data privacy. Especially in cases where there is limited available performance data overall, the accuracy of collaborators' performance models can be significantly improved through the use of shared synthetic training data samples. Further, the data synthesis has been shown to induce low computational overhead.

In the future, we will investigate alternative approaches to ensure privacy when sharing performance metrics of data analytics workloads. Moreover, we hope our short paper also inspires further research by others in the same direction.

## ACKNOWLEDGMENTS

This work has been supported through a grant by the German Research Foundation (DFG) as "C5" (grant 506529034).

## REFERENCES

[1] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2014. Machine Learning Classification over Encrypted Data. *Cryptology ePrint Archive* (2014).
[2] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).
[3] Haokun Fang and Quan Qian. 2021. Privacy Preserving Machine Learning with Homomorphic Encryption and Federated Learning. *Future Internet* 13, 4 (2021).
[4] Oded Goldreich. 1998. Secure Multi-Party Computation. *Manuscript. Preliminary version* 78, 110 (1998).
[5] Chin-Jung Hsu, Vivek Nair, Vincent W Freeh, and Tim Menzies. 2018. Arrow: Low-level Augmented Bayesian Optimization for Finding the Best Cloud VM. In *ICDCS '18*. IEEE.
[6] Muhammed Tawfiqul Islam, Shanika Karunasekera, and Rajkumar Buyya. 2021. Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments. *IEEE Transactions on Parallel and Distributed Systems* 33, 7 (2021).
[7] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine* 37, 3 (2020).
[8] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. 2021. When Machine Learning Meets Privacy: A Survey and Outlook. *ACM Computing Surveys* 54, 2 (2021).
[9] Haoyue Ping, Julia Stoyanovich, and Bill Howe. 2017. DataSynthesizer: Privacy-Preserving Synthetic Datasets. In *SSDBM '17*. ACM.
[10] Debbie Rankin, Michaela Black, Raymond Bond, Jonathan Wallace, Maurice Mulvenna, Gorka Epelde, et al. 2020. Reliability of Supervised Machine Learning Using Synthetic Data in Health Care: Model to Preserve Privacy for Data Sharing. *JMIR Medical Informatics* 8, 7 (2020).
[11] Dominik Scheinert, Alireza Alamgiralem, Jonathan Bader, Jonathan Will, Thorsten Wittkopp, and Lauritz Thamsen. 2021. On the Potential of Execution Traces for Batch Processing Workload Optimization in Public Clouds. In *Big Data '21*. IEEE.
[12] Dominik Scheinert, Philipp Wiesner, Thorsten Wittkopp, Lauritz Thamsen, Jonathan Will, and Odej Kao. 2023. Karasu: A Collaborative Approach to Efficient Cluster Configuration for Big Data Analytics. In *IPCCC '23*. IEEE.
[13] Navoda Senavirathne and Vicenç Torra. 2020. On the Role of Data Anonymization in Machine Learning Privacy. In *TrustCom '20*. IEEE.
[14] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-scale Advanced Analytics. In *NSDI '16*. USENIX.
[15] Jonathan Will, Lauritz Thamsen, Dominik Scheinert, Jonathan Bader, and Odej Kao. 2021. C3O: Collaborative Cluster Configuration Optimization for Distributed Data Processing in Public Clouds. In *IC2E '21*. IEEE.
[16] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. 2010. Spark: Cluster Computing with Working Sets. *HotCloud* 10, 10 (2010).

# Approximating Fork-Join Systems via Mixed Model Transformations

Rares-Andrei Dobre
Department of Computing
Imperial College London
London, UK
rares.dobre22@imperial.ac.uk

Zifeng Niu
Department of Computing
Imperial College London
London, UK
zifeng.niu19@imperial.ac.uk

Giuliano Casale
Department of Computing
Imperial College London
London, UK
g.casale@imperial.ac.uk

## ABSTRACT

While product-form queueing networks are effective in analyzing system performance, they encounter difficulties in scenarios involving internal concurrency. Moreover, the complexity introduced by synchronization delays challenges the accuracy of analytic methods. This paper proposes a novel approximation technique for closed fork-join systems, called MMT, which relies on transformation into a mixed queueing network model for computational analysis. The approach substitutes fork and join with a probabilistic router and a delay station, introducing auxiliary open job classes to capture the influence of parallel computation and synchronization delay on the performance of original job classes. Evaluation experiments show the higher accuracy of the proposed method in forecasting performance metrics compared to a classic method, the Heidelberger-Trivedi transformation. This suggests that our method could serve as a promising alternative in evaluating queueing networks that contains fork-join systems.

## CCS CONCEPTS

• **Software and its engineering → Software performance**; • **Networks → Network performance modeling**.

## KEYWORDS

Queueing Network, Fork-Join system, Synchronization Delay

**ACM Reference Format:**
Rares-Andrei Dobre, Zifeng Niu, and Giuliano Casale. 2024. Approximating Fork-Join Systems via Mixed Model Transformations. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3629527.3652277

## 1 INTRODUCTION

Modern software systems are typically implemented in a distributed manner to improve performance, reliability, and scalability [18]. Under this pattern, parallel and concurrent structures have gained increased importance over the years [9]. The software components are often parallelized to achieve higher execution efficiency and increase the utilization of the available resources. Parallel jobs typically follow a fork-join mechanism. A parallel job is forked into several tasks, which are executed concurrently on distinct resources within the system. After finishing the execution, a task has to await its sibling tasks at the join point. This job exits the join point and continues only when all its tasks have finished execution.

Queueing networks are a class of efficient performance models to understand the impact of execution mechanisms on system performance. A large number of computer systems can be abstracted as product-form queueing networks from which designers obtain accurate performance predictions [3]. Nevertheless, product-form queueing network models do not accommodate jobs that feature internal concurrency. Furthermore, the exact analysis for internal concurrency within a queueing network can rapidly lead to a state-space explosion.

In addition to the time spent on service, the total time of a parallel execution involves two other delays: queueing delay and synchronization delay [19]. Synchronization delay occurs on any completed task that waits for the completion of other sibling tasks before leaving the fork-join system. This coordination introduces dependencies and leads to an increased complexity in designing an accurate analysis for parallel executions. Therefore, it is necessary to have approximate analytic performance models to analyze parallelism for software designers and practitioners.

The technique developed by Heidelberger and Trivedi [13] is long established. This method, which we refer to as HT method in the rest of the paper, decomposes a job into a primary task and a fixed number of secondary tasks; these tasks are assumed to be independent of each other and belong to different job classes. Then probabilistic routing and pseudo-servers are used to replace parallel executions so that a product-form solution can be produced.

In this paper, we propose a novel approach, named MMT, for the analysis of fork-join systems. Similarly to the HT method, the fork and join are replaced by a probabilistic router and a delay, respectively. In our terminology, a probabilistic router is an abstract node that routes incoming jobs along output branches, according to a probabilistic routing policy and with zero service time. Beyond that, it introduces auxiliary open job classes to mimic the influence of parallel computation and synchronization delay on original job classes, which leads to a mixed queueing network model to analyze. The arrival rates and service rates of the auxiliary classes are computed from an iterative procedure that enables us to finally obtain the approximated solution of the original queueing network containing parallelism. The main performance measures in the fork-join system can be obtained using the method described.

The effectiveness of the proposed method is validated by comparison with simulations and the HT method. We use the simulation results as ground truth. Experiments are conducted on closed queueing networks with homogeneous or heterogeneous fork-join queues

This work is licensed under a Creative Commons Attribution International 4.0 License.

*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652277

**Table 1: Notations for considered queueing networks**

| Symbol | Definition |
|---|---|
| $r$ | index of the job class |
| $m$ | index of the service station |
| $c$ | index of the concurrent (fork-join) structure |
| $\mu_{m,r}$ | class-$r$ service rate at station $m$ |
| $\lambda_r$ | class-$r$ arrive rate (for open class) |
| $Q_{m,r}$ | class-$r$ queue length at station $m$ |
| $X_{m,r}$ | class-$r$ throughput at station $m$ |
| $T_{m,r}$ | class-$r$ response time at station $m$ |
| $U_{m,r}$ | class-$r$ utilization at station $m$ |
| $K$ | number of job classes in the network |
| $M$ | number of service stations in the network |
| $C$ | number of fork-join structures in the network |
| $F_c$ | number of parallel paths spawned by the fork of the structure $c$ |

(i.e., where servers can have the same or different service rates). Compared to the HT method, the proposed method offers lower error rates on predicted performance measures.

The rest of the paper is organized as follows. In Section 2, we provide background on queueing network theory, fork-join systems, and their analysis. In Section 3, we propose our MMT method for the analytic analysis of fork-join systems. In Section 4, we present evaluation experiments and results. In Section 5, we review related work. Finally, we conclude the paper in Section 6.

# 2 BACKGROUND

## 2.1 Queueing Networks

Queueing networks serve as a class of models for analyzing the performance of systems. They are made up of a collection of service stations indexed by $m = 1, \ldots, M$, in which jobs are queued and executed. A service station could have either infinite servers or finite servers, and is referred to as delay or queue station, respectively. The common use of delay stations is to represent the think times of workloads, while queue stations typically represent system resources [16]. In a queue station, there may be competition among jobs for the server, resulting in waiting times. A queueing network may execute multiple classes of jobs indexed by $r = 1, \ldots, K$. Distinct job classes typically feature different service rates $\mu_{m,r}$, and can be further categorized into closed and open classes. For closed job classes, a fixed number of jobs circulate within the network. For open job classes, jobs from a source continuously arrive to the network with rate $\lambda_r$. Scheduling policies determine the order in which jobs are served, with common policies including First-Come First-Serve (FCFS) and Processor Sharing (PS).

To evaluate the performance of a queueing network, one approach is to solve a system of global balance equations to get state probabilities of the underlying Markov chain where all performance measures can be further obtained. However, this method becomes hardly practical on complex networks due to the state-space explosion. Among queueing networks, there is a special class named product-form queueing networks. They have local balance properties and their exact performance measures can be obtained without resorting to the underlying state space. Mean Value Analysis (MVA) [21] is a prevalent technique to solve product-form queueing networks. In scenarios where exact analysis becomes computationally challenging, Approximate Mean Value Analysis (AMVA) [8] provides a practical alternative, offering insight into key performance measures such as queue lengths $Q_{m,r}$, throughputs $X_{m,r}$, response times $T_{m,r}$, and utilizations $U_{m,r}$.

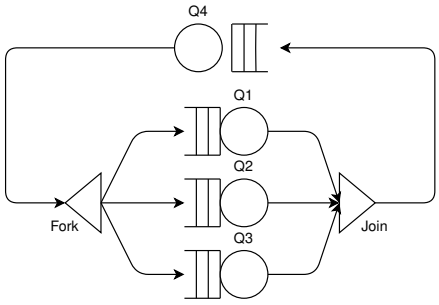



**Figure 1: Queueing network containing a fork-join system**

## 2.2 Fork-Join Systems

A queueing network may include a number of fork-join systems indexed by $c = 1, \ldots, C$. Fork-Join systems contain both fork and join nodes, which are a particular structure in queueing networks. The fork node has the property that any arriving job is split into multiple tasks to be serviced independently and in parallel, while the join node combines these tasks back into the original job after they have finished processing [23]. Let $F_c$ denote the number of parallel paths spawned by the fork node of the system $c$, the $F_c$ executions are assumed to be independent of each other so they do not intersect before reaching the join node. The notations for networks considered by this paper are summarized in Table 1.

An example of a closed queueing network containing a fork-join system is given in Figure 1. Jobs circulate between the fork-join system and a single queueing station. Any job that arrives at the fork node is split into three tasks to be processed on Q1, Q2, and Q3. After completing processing, they must wait for the sibling tasks to finish. Once all the tasks spawned by the job are finished, they are joined together back into the original job, which subsequently moves on to Q4.

## 2.3 Heidelberger-Trivedi Approximation Method

A queueing network that contains fork-join systems has no product-form solution [3], thus it cannot be efficiently solved in general. A classic transformation method is developed by [13], which originally transforms single-class closed queueing networks including a single parallel construct into a queueing network with no concurrency so that analytic methods such as AMVA can be applied. The main feature of this transformation is that each closed class is coupled with one auxiliary closed class for each parallel path.

The parallel constructs targeted by this work are analogous to fork-join systems. In its model, each job is represented by a primary task with multiple secondary tasks. Primary and secondary tasks are

used to represent the activity of the job executed outside and inside the fork-join system, respectively. Each primary task arriving at the fork node is forked into multiple secondary tasks, while secondary tasks arriving at the join node need to wait for their siblings to arrive before they can be joined back into the primary task.

This work associates the join node of the fork-join construct with a delay station. The time a primary task is supposed to spend at this delay station represents the overall response time of the job in the fork-join system, whereas for a secondary task it represents the corresponding synchronization delay. This approach also adds an auxiliary delay station to the network, which models the time a primary task spends outside the fork-join system. The secondary tasks use the auxiliary delay as their reference station. Their service rates are computed using an iterative procedure analogous to [14].

Suppose $D_0$ represents the response time of the fork-join system, $R_i$ denotes the response time at the $i$-th parallel path, and $F$ denotes the number of parallel paths spawned by fork $f$, the expectation of the job response time in the fork-join system is

$$E[D_0] = E[\max(R_1, \ldots, R_F)] \tag{1}$$

Therefore, the service time of the primary task at the delay station that replace the join node is $E[D_0]$. The $R_i$ are assumed to be exponential random variables with $E[R_i] = 1/\mu_i$ for all $i \in \{1, \ldots, F\}$, thus $E[D_0]$ can be further expressed as the following [24]

$$\begin{aligned} E[D_0] = &\sum_{i=1}^{F} \frac{1}{\mu_i} - \sum_{i<j} \frac{1}{\mu_i + \mu_j} + \sum_{i<j<k} \frac{1}{\mu_i + \mu_j + \mu_k} \\ &- \cdots + (-1)^{F-1} \sum_{i_1<\cdots<i_F} \frac{1}{\mu_{i_1} + \cdots + \mu_{i_F}} \end{aligned} \tag{2}$$

where $i, j, k, i_1, \ldots, i_F$ represent path indices.

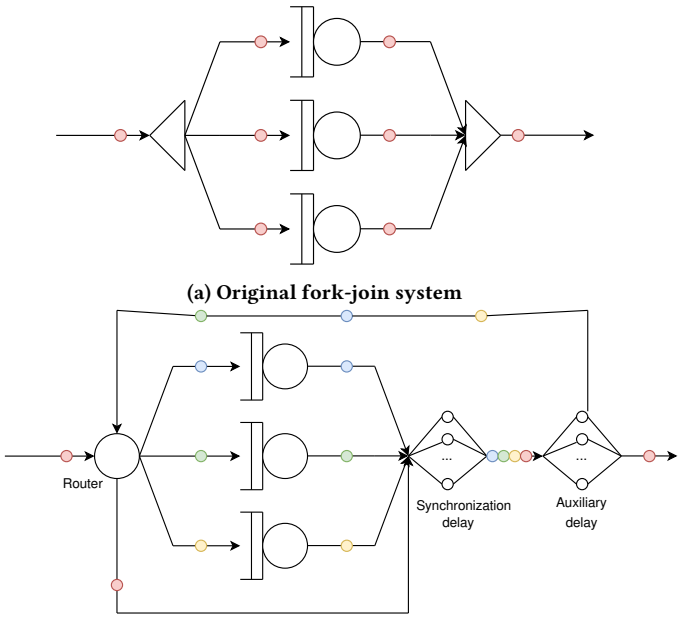
Let $D_i$ represent the service times of the secondary tasks at this delay station, i.e. the synchronization delays, $E[D_i]$ is obtained by subtracting the mean response time of the $i$-th parallel path from the fork-join response time

$$E[D_i] = E[D_0] - E[R_i] \tag{3}$$

Figure 2 depicts an example of a network alteration using the aforementioned procedure. The original job class and its primary tasks are depicted with red circles in the original and transformed systems, whereas the distinct secondary tasks are represented through blue, green, and yellow circles. It can be observed from Figure 2b that the HT method produces four job classes for this single class fork-join system with three parallel paths. In the transformed fork-join system, the primary tasks are routed straight to the synchronization delay station, whereas the secondary tasks are routed through the fork-join system. From the synchronization delay station, every job is routed to the auxiliary delay station. From the auxiliary delay station, the jobs belonging to the primary task continue proceeding through the rest of the network, while the jobs from the secondary tasks are sent out to the router. The obtained model is a product-form queueing network that can be solved easily.

## 3 PROPOSED METHODOLOGY

The proposed method is a simpler transformation of fork-join systems by introducing only one more delay station and fewer job




**(a) Original fork-join system**

**(b) Transforming original system into a network without concurrency**

**Figure 2: HT method for fork-join transformation**

**Table 2: Notations for auxiliary classes and network elements created by the MMT method**

| Symbol | Definition |
|---|---|
| $r$ | index of the original class |
| $r'_c$ | auxiliary class of $r$ created for the fork-join structure $c$ |
| $o_c$ | router created to replace the fork node of the fork-join structure $c$ |
| $d_c$ | delay station created to replace the join node of the fork-join structure $c$ |

classes. This transformation leads to a simple yet effective computation procedure for performance measures of the original system. The new notations introduced by the proposed method are summarized in Table 2.

### 3.1 Network Transformation

*3.1.1 Mixed model construction.* We transform the network into the one that can be applied to analytic approximations. The join node of the fork-join system $c$ is replaced by a delay station $d_c$, while the fork node is replaced by a router that forwards incoming jobs to original $F_c$ parallel paths. Each path carries an equal probability $\frac{1}{F_c}$ of being selected to forward a job to, and the sum of probabilities is 1. However, a router does not offer the functionality required to spawn other tasks. Thus, even if a job is routed to a parallel path, no sibling tasks are executed concurrently on the other parallel paths. To counter this issue, the parallelism induced by a fork-join system is simulated through the addition of open job classes, which we shall refer to as the auxiliary classes. The auxiliary classes

mirror the behavior of their original classes in terms of routing and service rates within the fork-join system. As shown in Figure 3a, the auxiliary classes are routed directly to the router from the source, and they are forwarded to the sink after leaving the delay station. Thus, their impact is limited to their corresponding fork-join systems.

In contrast to the HT method, each original class passing through the fork node is attributed only one auxiliary class whose jobs are meant to act as siblings of the original job class. Due to the approximation decision of assigning path selection probabilities equal to $\frac{1}{F_c}$, the transformation exercises all parallel paths equally. This is a key difference compared to HT, as it results in a model in which classes do not map in a one-to-one fashion with a particular parallel path. Hence, any approximation error that affects an auxiliary class is equally distributed across the paths. A comparison between both HT and MMT transformations is shown in Table 3, where the original queueing network has one fork-join system and $F$ represents the number of parallel paths.

*3.1.2 Arrival rate of the auxiliary open class.* Since we have introduced auxiliary open classes into the network, it is necessary to determine their arrival rates. In equilibrium, the router that replaces the fork node of the structure $c$ satisfies the flow balance condition [16]. Let $X_{o_c,r}$ denote the class-$r$ arrival rate/throughput at the router and assume $X_{o_c,r}$ to be the same as the class-$r$ arrival rate at the original fork node, the class-$r$ throughput at the original fork node is then $X_{o_c,r} \cdot F_c$. Therefore, the arrival rate of the auxiliary open class can be computed by balancing the flow at the router as

$$\lambda_{r'_c} = X_{o_c,r} \cdot (F_c - 1) \tag{4}$$

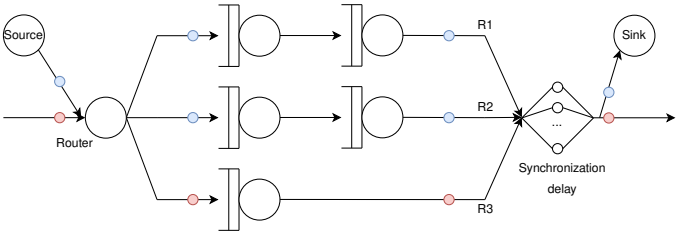
where $r$ denotes the original job class, $r'_c$ denotes the auxiliary class of $r$ created for the fork-join structure $c$, $\lambda_{r'_c}$ represents the arrival rate of the auxiliary open class, and $F_c$ here denotes the number of branches connected to the router.

## 3.2 Synchronization Delay

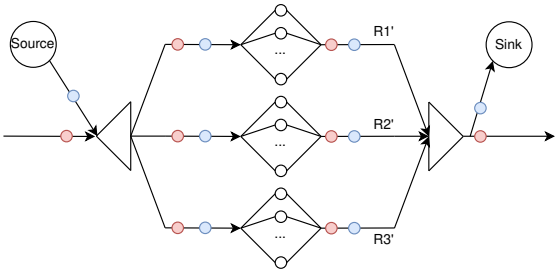
In this section, we illustrate how MMT method approximates the synchronization delay. For ease of presentation, we assume there is one fork-join system ($C = 1$) in the original network so we temporarily remove the subscript $c$. We assume that the response time at any parallel path $p$ to be approximately exponentially distributed, which is proved to an effective assumption for analytic analysis [22]. Besides, let random variables $R_{1,r}, \ldots, R_{F,r}$ represent class-$r$ response times at parallel paths $1, \ldots, F$, we assume they are mutually independent. Based on the same two assumptions as the HT method, we propose the following approximation approach consisting of two main steps to derive the synchronization delays for the fork-join system in a queueing network.

*3.2.1 Mean response time at a parallel path.* The first step of our approach is to obtain the response times at parallel paths of a fork-join system. For each path $p = 1, \ldots, F$, the mean response time is sum of mean response times at all queueing stations on that path.

We merge the performance measure of the auxiliary class with that of its corresponding original class. Given the merged queue length and throughput, Little's law [17] is then used to compute




**(a) The network gained from the transformation of a fork-join system**




**(b) Corresponding homogeneous fork-join system of the netwwork in Figure 3a**

**Figure 3: The proposed method for fork-join transformation**

the mean response time at each single station

$$R_{p,r} = \sum_{m \in M_p} \frac{Q_{m,r} + Q_{m,r'}}{X_{m,r} + X_{m,r'}} \tag{5}$$

where $M_p$ is the set of service stations on path $p$.

*3.2.2 Approximation by a homogeneous fork-join system.* The second step is to update the service rates of the original and auxiliary classes at the sychronization delay station.

We propose to approximate the given fork-join system with a homogeneous fork-join system. As shown in Figure 3b, the corresponding homogeneous fork-join system has the same number of parallel paths and features one delay station per path. The service times of both job classes at any delay station of the new fork-join systems are assumed to be exponential random variables with the means equal to the average of the original response times of the path executions

$$E[R'_{p,r}] = \frac{E[R_{1,r}] + \cdots + E[R_{F,r}]}{F}, \quad \forall p \in \{1, \ldots, F\} \tag{6}$$

where the random variable $R'_{p,r}$ represents the class-$r$ service time at the delay station of the parallel path $p$. The homogeneuous system can approximate the behavior of the orginal system since their response times are close to each other, i.e., $E[\max(R_{1,r}, \ldots, R_{F,r})] \approx E[\max(R'_{1,r}, \ldots, R'_{F,r})]$.

We therefore use $E[R'_{p,r}]$ to approximate the mean response time of each path in the original fork-join system, and then compute the synchronization delay by

$$E[D_{p,r}] = E[\max(R_{1,r}, \ldots, R_{F,r})] - E[R'_{p,r}], \quad \forall p \in \{1, \ldots, F\} \tag{7}$$

where $E[\max(R_{1,r}, \ldots, R_{F,r})]$ can be obtained by (2), and the random variable $D_{p,r}$ represents the class-$r$ synchronization delay at the parallel path $p$ of the original fork-join system.

**Table 3: Differences between two transformation methods**

| | HT | MMT |
|---|---|---|
| added elements | 2 delays | 1 delay, 1 source |
| network type | closed | mixed |
| number of classes | $K(1+F)$ | $2K$ |
| synchronization delay | heterogeneous | homogeneous (after aggregation) |

**Algorithm 1** Procedures to solve original queueing networks

**Input:** a queueing network, $tol$
**Output:** $Q_{m,r}, X_{m,r}, T_{m,r}, U_{m,r}$

1: initialize $X_{o,r} \leftarrow 0$, $Q_{m,r} \leftarrow 1$, $Q^0_{m,r} \leftarrow 1$
2: $stop \leftarrow false$
3: obtain $\boldsymbol{Q}$ by fork-join transformation
4: **while** $stop$ == false **do**
5: **if** $\max\left(1\text{-}(Q^0_{m,r}/\, Q_{m,r})\right) < tol$ **then**
6: $stop \leftarrow true$
7: **else**
8: $Q^0_{m,r} \leftarrow Q_{m,r}$
9: **end if**
10: $Q_{m,r}, Q_{m,r'}, X_{m,r}\, X_{m,r'}, T_{m,r}, T_{m,r'}, U_{m,r}, U_{m,r'} \leftarrow \text{AMVA}(\boldsymbol{Q})$
11: **for** $r = 1, \ldots, K$ **do**
12: $X_{o,r} \leftarrow \text{Avg}(X_{o,r}, X_{d,r})$
13: $\lambda_{r'} \leftarrow X_{o,r} \cdot (F-1)$
14: obtain $R_{1,r}, \ldots, R_{F,r}$ by (5)
15: obtain $\mathbb{E}[D_0]$ by (2)
16: update $\mu_{d,r}$ and $\mu_{d,r'}$ by (7)
17: $Q_{m,r} \leftarrow Q_{m,r} + Q_{m,r'}$; $X_{m,r} \leftarrow X_{m,r} + X_{m,r'}$; $T_{m,r} \leftarrow Q_{m,r}/X_{m,r}$; $U_{m,r} \leftarrow U_{m,r} + U_{m,r'}$
18: **end for**
19: **end while**

## 3.3 Algorithm

In our method, the arrival rates of auxiliary classes and the service rates at the delay server are not known in advance. This implies that an iterative computation framework is needed to approximate the solution of the original network.

The framework is shown in Algorithm 1. For ease of presentation, we consider one fork-join system and temporarily remove the subscript $c$. The input of the algorithm includes an original queueing network, a tolerance $tol$ that serves as the iteration stopping threshold, and a boolean value $stop$ that determines whether to stop the iteration. We set the initial value of $X_{o,r}$ to 0, and both $Q_{m,r}$, $Q^0_{m,r}$ to the same number (line 1), and transform the original queueing network into a product-form mixed queueing network by the proposed procedure (line 3). The stopping criterion is the difference between the queue lengths of two successive iterations (lines 5-9). At each iteration, the transformed queueing network is solved by AMVA (line 10). Then, for each auxiliary open class, we update its arrive rate (lines 12-13) and update the service rates of both auxiliary and corresponding original classes at the synchronization delay station (lines 14-16). In this way, the parameters of the queueing network are updated. The last step of an iteration is to merge the results for each original class (line 17).

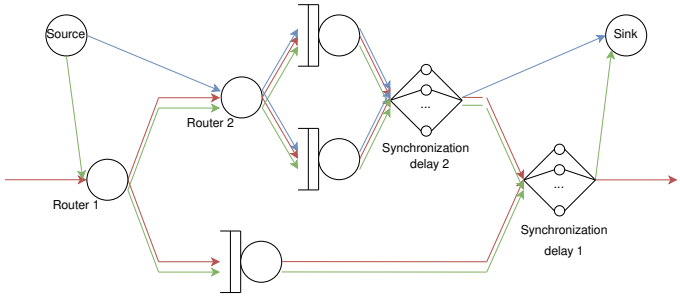



**Figure 4: An example of nested fork-join transformation by the proposed method**

## 3.4 Extend Method to Nested Fork-Join System

A nested fork-join structure is a hierarchical arrangement of fork-join systems, utilizing nested fork-join structures holds significance in the field of software development [4, 15]. Figure 4 shows a network transformed from a nested fork-join system by the proposed procedure. As it can be observed, the original system includes two fork-join structures. We refer to the outer fork-join as $FJ_1$ and to the inner fork-join as $FJ_2$. In this network, the possible paths of the original job class are depicted in red, whereas the possible paths of its auxiliary job classes are depicted in green or blue. The green paths are for the auxiliary class created for $FJ_1$, whereas the blue color denotes the paths of the auxiliary class created for $FJ_2$.

To adapt to nested fork-join systems, for each original class, we first build auxiliary classes for every fork-join structure visited by the original class. Two additions are introduced in the MMT method compared to our original method for systems without nested fork-join structures. The first addition is to start computing the synchronization delay only at the outermost fork-join structure, and then compute the nested fork-join structures recursively.

The second addition is the calculation of arrival rates of the auxiliary classes. Because nested systems incorporate inner fork-join structures and auxiliary classes are created for each of them, solely considering the throughput of the original class at an inner fork to compute the arrival rate of the auxiliary class associated to this structure is not enough. The auxiliary class created for an inner fork-join structure simulates the sibling tasks executing concurrently of the auxiliary classes created for the outer fork-join structures. In other words, the influence of both the original class and the outer auxiliary classes should be considered. Hence, before computing the arrival rate, the throughput $X_{d_c,r}$ is updated using throughputs of class-$r$ and its auxiliary classes at that delay station

$$X_{d_c,r} = X_{d_c,r} - X_{d_c,r'_c} + \sum_{s=1}^{C} X_{d_c,r'_s} \tag{8}$$

where $C$ denotes the set of fork-join structures in the queueing network, $r'_c$ denotes the auxiliary class of $r$ created for the fork-join structure $c$, and $d_c$ denotes the delay station created to replace the join node of the fork-join structure $c$.

We refer to class-$r$ and its auxiliary classes as *$r$-related classes*. Equation (8) merges the throughputs of all $r$-related job classes at the delay station, except $X_{d_c,r'_c}$, which is the throughput of the auxiliary class created for the fork-join structure associated to the

**Table 4: Distinct fork-join queues used in evaluation**

| Groups | Topology |
|---|---|
| heterogeneous_FCFS_1 | $< Q_1 \parallel Q_2 >$ |
| heterogeneous_FCFS_2 | $< Q_1 \parallel Q_2 \parallel Q_3 >$ |
| heterogeneous_FCFS_3 | $< Q_1 \parallel (Q_2 \rightarrow Q_3) >$ |
| heterogeneous_PS_1 | $< Q_1 \parallel Q_2 >$ |
| heterogeneous_PS_2 | $< Q_1 \parallel Q_2 \parallel Q_3 >$ |
| heterogeneous_PS_3 | $< Q_1 \parallel (Q_2 \rightarrow Q_3) >$ |
| heterogeneous_PS_4 | $< Q_1 \parallel Q_2 >$ |
| heterogeneous_PS_5 | $< Q_1 \parallel Q_2 \parallel Q_3 >$ |
| homogeneous_FCFS_1 | $< Q_1 \parallel Q_2 >$ |
| homogeneous_FCFS_2 | $< Q_1 \parallel Q_2 \parallel Q_3 >$ |
| homogeneous_PS_1 | $< Q_1 \parallel Q_2 >$ |
| homogeneous_PS_2 | $< Q_1 \parallel Q_2 \parallel Q_3 >$ |
| homogeneous_PS_3 | $< Q_1 \parallel Q_2 >$ |
| homogeneous_PS_4 | $< Q_1 \parallel Q_2 \parallel Q_3 >$ |

current delay station $d_c$. In other words, this equation gives the total $r$-related throughputs that do not leave for the sink from the current delay station. If the fork-join structure $c$ is not nested (i.e., the outermost fork-join), $X_{d_c,r'_c}$ will remain unchanged.

## 4 EVALUATION

We first compare the accuracy of the HT method [13] and MMT method against simulation results obtained by Java Modeling Tool (JMT) [2]. The implementations of these methods are included by LINE [5] that is a algorithmic framework for queueing networks and layered queueing networks [6]. The involved closed queueing networks in our evaluation can be categorised into distinct groups depending on the service rates of parallel executions (homogeneous or heterogeneous) and the scheduling used at the queueing stations (FCFS or PS).

There are two job classes in every queueing network. Both classes are closed with a population of 10 jobs each. The queueing networks always include a fork-join system containing two or three queueing stations. The service rates at these stations are randomly generated, with the average service time between 0.3 and 0.8. Table 4 shows a list of the fork-join queues used in the first experiments. We use the following notations to describe their topology: < denotes a fork node, > represents a join node, || defines a parallel branch, and → defines a serial routing.

The evaluation results are shown in Table 5. It can be observed that the proposed method achieves lower errors on most cases than the HT method. Compared to the baseline, the MMT method reduces the prediction error of queue length, response time, utilization, and throughput by 30.9%, 62.8%, 34.6%, and 35.3% on average. Figure 5a demonstrates that the two methods exhibit similar prediction accuracy on homogeneous networks, whereas Figure 5c illustrates that our method notably achieves higher accuracy on heterogeneous networks. Apart from accuracy, runtime is the other important factor to consider. As shown in Figure 5b and 5d, the average runtime of our method is less than 0.015s, substantially lower than that of the HT method, which is around 0.03s, as the runtime of the AMVA scales with the increasing number of service

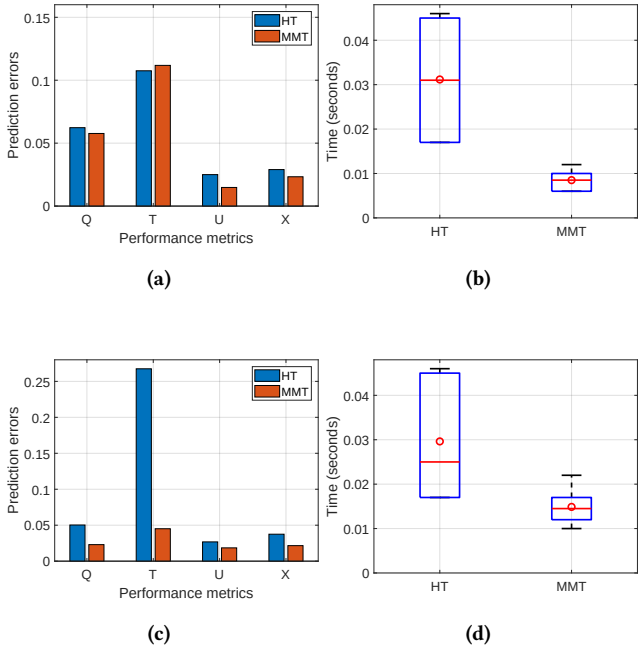



**Figure 5: (a)-(b) for homogeneous networks. (c)-(d) for heterogeneous networks. (a),(c): Average prediction errors by HT and MMT methods. (b),(d): Box plots of average runtimes for both methods. Red circles and lines inside boxes represent mean and median values, respectively.**

stations and job classes. Compared to our method, where only one auxiliary job class is created for a fork-join system, the HT method creates one auxiliary class for each parallel path of the concurrent system and an additional delay station to model the time spent outside the fork-join system. Hence, the models created by our method are more efficient to compute compared to those created by the HT method.

We then evaluate the MMT method on nested fork-join queues. Here we only compare the results with that of simulations since the baseline HT method is not designed for nested fork-join queues. The networks used for evaluation and numerical results are provided in Table 6 and Table 7, respectively. Figure 6 visualizes mean and maximum errors in FCFS and PS groups. As can be observed, the MMT method provides accurate predictions. This method inherently possesses the capability to handle nested fork-join systems because the auxiliary classes it creates are restricted to their corresponding fork-join systems, and they are independent of the time their original classes spend outside the fork-join systems, which is a distinct advantage of our approach. In contrast, the HT method is initially devised for a network with a single fork-join system.

## 5 RELATED WORK

Fork-Join queueing networks represent the key to modelling and solving parallel systems. However, they do not follow the product-form restrictions, so most algorithms devised for fork-join networks

**Table 5: Queue length, response time, utilization and throughput errors of HT and MMT methods**

| Groups | ErrQ | | ErrT | | ErrU | | ErrX | |
|---|---|---|---|---|---|---|---|---|
| | HT | MMT | HT | MMT | HT | MMT | HT | MMT |
| heterogeneous_FCFS_1 | 0.034 | **0.016** | 0.226 | **0.022** | **0.002** | 0.003 | **0.004** | 0.007 |
| heterogeneous_FCFS_2 | 0.061 | **0.029** | 0.412 | **0.041** | **0.004** | 0.018 | **0.003** | 0.021 |
| heterogeneous_FCFS_3 | 0.066 | **0.024** | 0.409 | **0.034** | 0.043 | **0.010** | 0.070 | **0.012** |
| heterogeneous_PS_1 | 0.063 | **0.014** | 0.262 | **0.031** | 0.010 | **0.009** | 0.016 | **0.010** |
| heterogeneous_PS_2 | 0.025 | **0.024** | 0.140 | **0.070** | **0.025** | 0.046 | **0.027** | 0.046 |
| heterogeneous_PS_3 | 0.067 | **0.019** | 0.278 | **0.046** | 0.086 | **0.031** | 0.122 | **0.039** |
| heterogeneous_PS_4 | 0.049 | **0.042** | **0.070** | 0.087 | 0.040 | **0.021** | 0.053 | **0.022** |
| heterogeneous_PS_5 | 0.037 | **0.016** | 0.343 | **0.030** | **0.004** | 0.010 | **0.005** | 0.016 |
| homogeneous_FCFS_1 | **0.069** | 0.078 | 0.125 | **0.116** | 0.013 | **0.009** | 0.021 | **0.018** |
| homogeneous_FCFS_2 | **0.083** | 0.094 | 0.114 | **0.110** | 0.035 | **0.018** | 0.039 | **0.032** |
| homogeneous_PS_1 | 0.041 | **0.032** | **0.053** | 0.074 | 0.014 | **0.010** | 0.021 | **0.017** |
| homogeneous_PS_2 | 0.062 | **0.040** | 0.124 | **0.116** | 0.039 | **0.015** | **0.025** | 0.028 |
| homogeneous_PS_3 | 0.053 | **0.045** | **0.073** | 0.099 | 0.025 | **0.014** | 0.027 | **0.016** |
| homogeneous_PS_4 | 0.066 | **0.057** | 0.156 | 0.156 | 0.024 | **0.023** | 0.041 | **0.029** |
| Mean | 0.055 | **0.038** | 0.199 | **0.074** | 0.026 | **0.017** | 0.034 | **0.022** |

**Table 6: Nested fork-join queues used in evaluation**

| Groups | Topology |
|---|---|
| FCFS_1 | $< Q_1 \mid\mid (< Q_2 \mid\mid Q_3 >) >$ |
| FCFS_2 | $< Q_1 \mid\mid (< Q_2 \mid\mid Q_3 >) >$ |
| FCFS_3 | $< (< Q_1 \mid\mid Q_2 >) \mid\mid (< Q_3 \mid\mid Q_4 >) >$ |
| FCFS_4 | $< (< Q_1 \mid\mid Q_2 >) \mid\mid (< Q_3 \mid\mid Q_4 >) >$ |
| PS_1 | $< Q_1 \mid\mid (< Q_2 \mid\mid Q_3 >) >$ |
| PS_2 | $< Q_1 \mid\mid (< Q_2 \mid\mid Q_3 >) >$ |
| PS_3 | $< (< Q_1 \mid\mid Q_2 >) \mid\mid (< Q_3 \mid\mid Q_4 >) >$ |
| PS_4 | $< (< Q_1 \mid\mid Q_2 >) \mid\mid (< Q_3 \mid\mid Q_4 >) >$ |

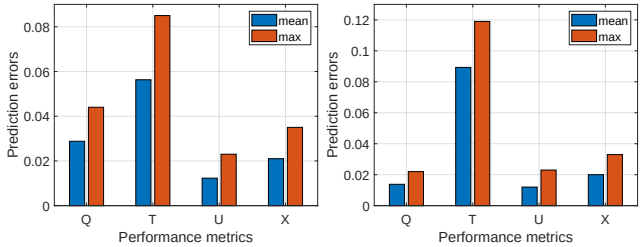



**Figure 6: (a) for networks with FCFS scheduling. (b) for networks with PS scheduling.**

**Table 7: Queue length, response time, utilization and throughput errors of the MMT method**

| Groups | ErrQ | ErrT | ErrU | ErrX |
|---|---|---|---|---|
| FCFS_1 | 0.029 | 0.035 | 0.004 | 0.009 |
| FCFS_2 | 0.023 | 0.036 | 0.004 | 0.007 |
| FCFS_3 | 0.044 | 0.069 | 0.018 | 0.035 |
| FCFS_4 | 0.019 | 0.085 | 0.023 | 0.033 |
| PS_1 | 0.011 | 0.085 | 0.004 | 0.008 |
| PS_2 | 0.012 | 0.070 | 0.004 | 0.007 |
| PS_3 | 0.022 | 0.119 | 0.017 | 0.032 |
| PS_4 | 0.010 | 0.083 | 0.023 | 0.033 |
| Mean | 0.021 | 0.073 | 0.012 | 0.021 |

are approximations and bounds, see e.g. [23],[7]. The only exact solution has been devised for two parallel servers [20]. The main difficulty in analysing fork-join queueing networks stems from the synchronization delays incurred by the jobs waiting for the other jobs created by a job to finish [11].

Duda and Czachórski [11] devise an algorithm to analyse fork-join queueing networks by replacing the fork-join constructs with load-dependent queueing stations. Its foundation consists of the flow-equivalent server method [3] and the decomposition principle. Varki [25] modifies the computation of residence time in the MVA algorithm to adapt to the closed, single-class queueing networks containing fork-join systems. This modification assumes service stations to have exponentially distributed service times and FCFS scheduling strategies. Alomari and Menasce [1] propose a method for analyzing fork-join systems involving servers with heterogeneous service times in open networks. The core of this method involves establishing bounds on the response time of a job in a fork-join system. This is achieved by analyzing the system under two scenarios: one where all stations have the same service rates and the other where their service rates vary. Mak and Lundstrom [19] introduce an iterative algorithm capable of approximating the performance measures of directed acyclic graphs abstracted from parallel systems in polynomial space and time. Franks and Woodside [12] illustrate the capability of the layered modelling framework in which a platform for defining models with parallelism can be conveniently used.

## 6 CONCLUSION

The paper proposes an accurate and computationally efficient approach for analyzing closed queueing networks containing fork-join systems. The core of this method involves establishing auxiliary open job classes to simulate the behavior of the original parallelism. This transformation leads to a mixed queueing network model that can be solved by analytic method. Compared to the well-established Heidelberger-Trivedi method, which uses an one-to-one fashion to create auxiliary closed job classes for each parallel path, our approach produces one auxiliary open job classes for the entire fork-join systems. The evaluation results show that our method achieves lower error rates. Meanwhile, the proposed method is faster than the baseline method since our transformed network has less number of job classes that requires less analytic computations. In addition, the design of our transformation enables us to deal with nested fork-join system, which is a notable advantage. Hence, this paper contributes a simple yet effective fork-join transformation which has the potential to be of great value in areas of the concurrent system research. An extended version of the work presented in this paper is available in [10].

## REFERENCES


[1] Firas Alomari and Daniel A Menasce. 2013. Efficient response time approximations for multiclass fork and join queues in open and closed queuing networks. *IEEE Transactions on Parallel and Distributed Systems* 25, 6 (2013), 1437–1446.

[2] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. 2009. JMT: performance engineering tools for system modeling. *ACM SIGMETRICS Performance Evaluation Review* 36, 4 (2009), 10–15.

[3] Gunter Bolch, Stefan Greiner, Hermann De Meer, and Kishor S Trivedi. 2006. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications.* John Wiley & Sons.

[4] Michael Burke, Ron Cytron, Jeanne Ferrante, and Wilson Hsieh. 1989. Automatic generation of nested, fork-join parallelism. *The Journal of Supercomputing* 3 (1989), 71–88.

[5] Giuliano Casale. 2020. Integrated performance evaluation of extended queueing network models with line. In *2020 Winter Simulation Conference (WSC)*. IEEE, 2377–2388.

[6] Giuliano Casale, Yicheng Gao, Zifeng Niu, and Lulai Zhu. 2023. LN: A Flexible Algorithmic Framework for Layered Queueing Network Analysis. *ACM Transactions on Modeling and Computer Simulation* (2023).

[7] Giuliano Casale, Richard Muntz, and Giuseppe Serazzi. 2008. Geometric bounds: A noniterative analysis technique for closed queueing networks. *IEEE Trans. Comput.* 57, 6 (2008), 780–794.

[8] K Mani Chandy and Doug Neuse. 1982. Linearizer: A heuristic algorithm for queueing network models of computing systems. *Commun. ACM* 25, 2 (1982), 126–134.

[9] David Culler, Jaswinder Pal Singh, and Anoop Gupta. 1999. *Parallel computer architecture: a hardware/software approach.* Gulf Professional Publishing.

[10] Rares-Andrei Dobre. 2023. *Stochastic Modelling in JLINE: Redesigning and Augmenting the MVA Solver with Fork-Join Analysis Methods.* Technical Report. MSc Final project, Department of Computing, Imperial College London.

[11] Andrzej Duda and Tadeusz Czachórski. 1987. Performance evaluation of fork and join synchronization primitives. *Acta Informatica* 24 (1987), 525–553.

[12] Greg Franks and Murray Woodside. 1998. Performance of multi-level client-server systems with parallel service operations. In *Proceedings of the 1st international workshop on Software and performance*. 120–130.

[13] Heidelberger and Trivedi. 1983. Analytic queueing models for programs with internal concurrency. *IEEE Trans. Comput.* 100, 1 (1983), 73–82.

[14] Patricia A Jacobson and Edward D Lazowska. 1982. Analyzing queueing networks with simultaneous resource possession. *Commun. ACM* 25, 2 (1982), 142–151.

[15] Gokcen Kestor, Sriram Krishnamoorthy, and Wenjing Ma. 2017. Localized fault recovery for nested fork-join programs. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 397–408.

[16] Edward D Lazowska, John Zahorjan, G Scott Graham, and Kenneth C Sevcik. 1984. *Quantitative system performance: computer system analysis using queueing network models.* Prentice-Hall, Inc.

[17] John DC Little. 1961. A proof for the queuing formula: L= $\lambda$ W. *Operations research* 9, 3 (1961), 383–387.

[18] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. 1995. Specifying distributed software architectures. In *Software Engineering—ESEC'95: 5th European Software Engineering Conference Sitges, Spain, September 25–28, 1995 Proceedings 5*. Springer, 137–153.

[19] Victor W Mak and Stephen F. Lundstrom. 1990. Predicting performance of parallel computations. *IEEE Transactions on Parallel & Distributed Systems* 1, 03 (1990), 257–270.

[20] Randolph Nelson and Asser N Tantawi. 1988. Approximate analysis of fork/join synchronization in parallel queues. *IEEE transactions on computers* 37, 6 (1988), 739–743.

[21] Martin Reiser and Stephen S Lavenberg. 1980. Mean-value analysis of closed multichain queuing networks. *Journal of the ACM (JACM)* 27, 2 (1980), 313–322.

[22] S Salza and SS Lavenberg. 1981. Approximating response time distributions in closed queueing network models of computer performance. (1981).

[23] Alexander Thomasian. 2014. Analysis of fork/join and related queueing systems. *ACM Computing Surveys (CSUR)* 47, 2 (2014), 1–71.

[24] Kishor S Trivedi. 2008. *Probability & statistics with reliability, queuing and computer science applications.* John Wiley & Sons.

[25] Elizabeth Varki. 1999. Mean value technique for closed fork-join networks. *ACM SIGMETRICS Performance Evaluation Review* 27, 1 (1999), 103–112.

# Establish a Performance Engineering Culture in Organizations

## Performance as a Value

Josef Mayrhofer
Performetriks LLC
New Jersey / USA
Josef@performetriks.com

## ABSTRACT

Performance Engineering still needs to be adopted in many organizations. At the same time, user expectations for fast and reliable applications are increasing. This paper discusses different approaches to establishing a performance engineering culture. After highlighting some of the challenges that hold businesses back from making performance a shared responsibility, we convey a success story about how performance became a matter for everyone in a large European bank.

## CCS CONCEPTS

•Software and its engineering, Software and its engineering~Software maintenance tools, Software and its engineering~Application specific development environments, Software and its engineering~Software as a service orchestration system, Software and its engineering~Object oriented frameworks

## KEYWORDS

Performance Engineering, Culture, Performance Testing, Performance Monitoring, Performance Touchpoints, Performance as a value, Reliability, Resilience, Quality, Gobenchmark

**ACM Reference format:**

Josef Mayrhofer. 2024. Establish a Performance Engineering Culture in Organizations: Performance as a Value, In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering, May 7–11, 2024, London, United Kingdom*, ACM, New York, NY, USA. 7 pages, https://doi.org/10.1145/3629527.3652278

## 1 WHAT IS PERFORMANCE ENGINEERING

Performance Engineering combines all techniques to design, build, and operate IT services with performance in mind.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ICPE '24 Companion, May 7–11, 2024, London, United Kingdom
© 2024 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652278
.

It goes far beyond validating performance requirements and involves performance monitoring. Some typical performance engineering activities are:

- Performance requirement review and specification
- Code and design review
- Workload modeling
- Design and implement performance tests
- Execute and analysis of performance tests
- Performance defect tracking and reporting
- Performance troubleshooting support and tuning
- Performance tracing and monitoring
- Performance audit and feedback

Performance [4] is a pervasive quality of software systems; everything affects it. Performance risks increase the later we start following performance engineering practices or the earlier we stop applying them. Performance engineering is a continuous process that is very similar to software engineering. We can only stop software engineering activities by setting the software end of life. Similarly, if we fail to adopt or stop performance engineering activities, it is only a matter of time before severe reliability problems disrupt our operations.

## 2 ROLES AND RESPONSIBILITIES

We have several performance engineering roles depending on the size of a project or organization. The performance engineer is technically gifted, a hands-on software developer aware of the latest advancements and performance engineering tools. When it comes to architecture reviews, layout of performance requirements, or design of performance validation and monitoring approach, the performance architect plays a crucial role. The performance manager drives the communication and reporting activities to keep everyone on Board with the performance development efforts.

In smaller organizations, a performance engineer usually combines all three roles, while large enterprises split these roles among multiple teams.

## 3 PERFORMANCE ENGINEERING TOUCHPOINTS

### 3.1 Organizational Touchpoints

From an organizational perspective, we see several approaches to how enterprises implement performance engineering activities.

#### 3.1.1 Project-based

Back in 1999, performance engineering was a project-related effort. I remember a project at an e-commerce business in Germany. A multi-million-dollar project was launched, and the team faced severe slowness issues during checkout a few weeks before the scheduled launch date. The project lead hired me as their performance expert, equipped me with a load testing tool, and assigned the load testing task. After implementing and executing a few fundamental load tests, I had all the evidence to demonstrate that the new shop could not handle concurrent user traffic. This was no good news because it took several weeks for the developers to fix these issues, and the organization had to postpone their product launches.

#### 3.1.2 Center of Excellence

In the late 90s, organizations streamlined their processes and wiped-out redundancies. The so-called CoE or Center of Excellence optimizes [5] each value component. A CoE performance owns all processes, methods, and tools related to performance engineering. They provided coaching for project teams, maintained the load testing tools, and, in some cases also, supported test implementation and execution.

#### 3.1.3 As a Service

The as-a-service economy has become popular during the Industry 4.0 [8] movement. Driven by cost transparency and efficiency, Performance engineering as a service is often provided by managed service providers or internal teams. They do everything from requirement gathering to project-based load test implementation, execution, and reporting. This model has benefits, such as buying what you need, but disadvantages, such as not going the extra mile when required and slightly higher costs.

#### 3.1.4 Performance as a Value

For performance to be valuable, it must grow and endure. If we look at software development and bug-fixing efforts, we understand that late defect discovery results in high defect costs, delays releases, and negatively impacts end users. By fixing such late-discovered performance issues, you can only heal something already broken but won't create endurable value.

The value of performance requires initial and continuous investments. Like an investor who puts their money into promising assets, the "Performance as a Value" technique follows a risk-based approach. There is no reason to put the same performance investment in all your business applications. Instead, we run a risk rating on applications and their changes, and depending on this rating, we implement mitigating measures. This risk mitigation applies to pre-production performance activities and includes performance monitoring, alerting, and tracing on production.

### 3.2 Life Cycle Touchpoints

From a software development process perspective, there are several approaches to how enterprises integrate performance engineering activities. The chess grandmaster [6] put it very well when he said, "To improve your game, you must study the endgame before everything else, for whereas the endings can be studied and mastered by themselves, the middle game and the opening must be studied in relation to the endgame." This end-game thinking is often the secret to a successful performance engineering project because we put ourselves, in the end, user's shoes and design a performance approach to validate the system requirements under a realistic production-like setting.

Performance requires early and continuous efforts to keep it at the needed level. The researcher Capers Jones pointed out that defect costs are low in the early stages [1] but up to 640 times higher if discovered in production. In Figure 1, we highlight the dependencies between introducing and resolving defects. When we bring defect resolution closer to coding, the costs could be reduced. At the same time, reputation and business risks increase when performance problems are found late or, in the worst case, in production. Performance must be part of the entire software development lifecycle.

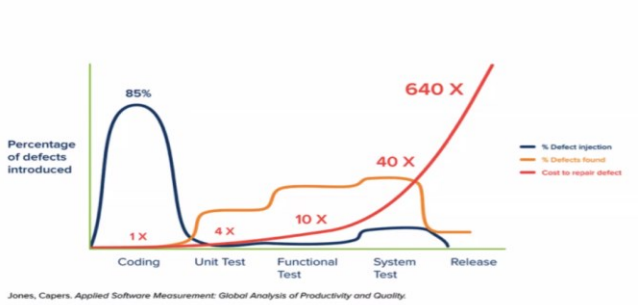



**Fig. 1.** Statistic from Applied Software Measurement [1] to demonstrate the dependency between defect introduction and defect resolution on software development costs.

#### 3.2.1 Design for Performance

Technology alone is not a guarantee of success [2]. Jim Collins explained the role of technology in successful businesses. It seems the same is true for building fast and reliable business applications. We should not hope that the latest technologies guarantee reliable IT services. Instead, we must lay out realistic performance requirements and make them part of our early software design decisions.

#### 3.2.2 Coding for Performance

When developers know performance requirements such as request volumes or response time expectations, they can integrate lazy loading, resilience, or caching concepts in their software components. At the same time, developers should be motivated to implement unit tests to validate the performance as part of their build processes. The benefit of these practices is that they identify and fix performance problems earlier and at a lower effort.

#### 3.2.3 Testing for Performance

Performance must be validated, release by release. Any change comes with a performance risk. We can ignore or mitigate these

risks by running a predefined set of performance tests and validating our performance quality gates. This continuous performance validation creates a quality-first mindset and avoids expensive surprises when we deploy our new features to the user community.

These days, performance validations are highly automated and fully integrated into the deployment process. When performance testing is a lean and continuous process, we detect problems much earlier. Ideally, performance testing-related activities are scalable self-service, and we have performance-trending capabilities and dashboards in place. The chart in Figure 2 outlines how continuous performance testing supports early component-level performance tests as well as integration and end-to-end performance tests.

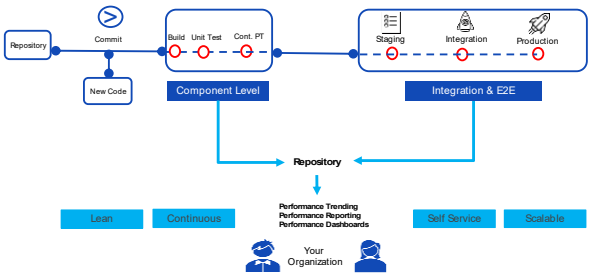



**Fig. 2.** A visualization of how performance testing can become a continuous process.

#### 3.2.4 Operations for Performance

In our digitized world, IT services must sustain significant variations in user and data volumes. The expectations for fast and reliable business services grow, and when customers get frustrated due to performance issues, we see a loss in sales. It's more important than ever to identify the root cause of degradations and implement the remediations fast by keeping the mean time to repair (MTTR) low.

## 4 ESTABLISH A PERFORMANCE ENGINEERING CULTURE IN A LARGE BANK IN EUROPE

In 2019, performance engineering was absent from a large European bank; their customers complained, and regulatory agencies were on the doorstep to review how this bank ensures that only validated software is deployed to production. The Board launched a program to modernize their testing activities, including identifying and remedying gaps. Within a few days, it was evident that performance engineering was one of their challenges, and the team hired me to make performance a matter for everyone. In this section, I explain how we've established a culture of performance discipline in this organization.

### 4.1 Culture of Performance Discipline

Performance requires awareness and commitment from management from the first steps to the entire length of this investment. We tried to get everyone settled in and explained why, what, and how we planned to implement performance engineering. The following sections outline how we've implemented performance as a shared responsibility. From leadership to business and technical roles, performance became a matter for everyone.

#### 4.1.1 Performance for Leadership

Empowered by line managers, the QA, Project, and performance lead conducted performance risk assessments. Depending on the outcome of these risk ratings, they engaged performance experts to validate their performance requirements.

The leadership team's role is to remind everyone involved in the software development process about the importance of reliability and performance. Any enterprise application changes are on the daily agenda. The leadership team provides the framework and rules for performance risk assessments. Supported by a Performance expert, they can review their risk assessment and plan meaningful performance tests to mitigate identified performance risks.


**Fig. 3.** The Leadership's roles and responsibilities to make performance a continuous process.

#### 4.1.2 Performance for Business

Performance requires end-game thinking, so we've made business service owners, business QA, and testers responsible for performance requirements and their validation. The transformation at the business level, from what to build to how to build it, created much better awareness for performance considerations.

The focus changed from testing functionality to testing how the end customers will use the product. For each release performance requirement, risk assessment and load and performance testing became a fundamental, planned discipline, as outlined in Figure 4.

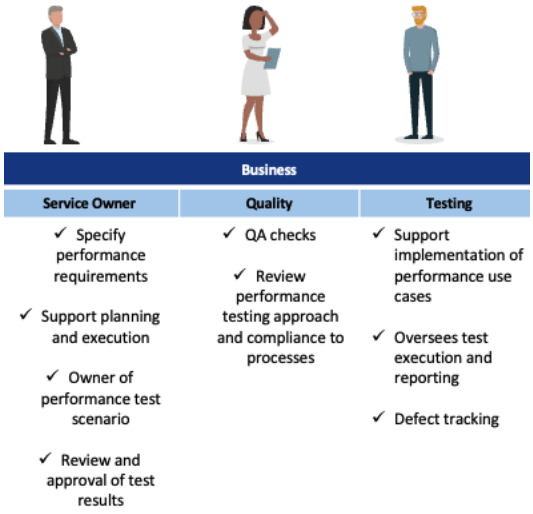



**Fig. 4.** The business teams' roles and responsibilities to make performance a continuous process.

#### 4.1.3 Performance for Technical Experts

Business teams got support from technical services owners and developers. The technical service owner had a shared responsibility for performance, as outlined in Figure 5, and worked with developers on fixing identified performance issues. Once we've created this common understanding for performance, we looked at the processes in this bank. We intended to make performance a continuous activity without building large additional teams.

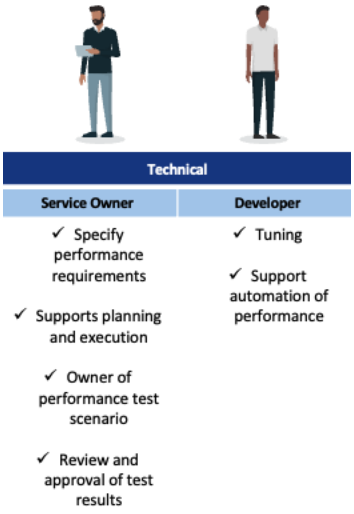



**Fig. 5.** The technical teams' roles and responsibilities to make performance a continuous process.

During the first few months, we focused on building a small core team for performance engineering. This self-motivated team designed a performance engineering framework as laid out in the performance as a value section 3.1.4 to clarify a few general rules, such as who is in charge, what the rules are, and how to validate the performance requirements. At the same time, we deployed a test lab, installed performance monitoring and load injection tools, and educated the teams on how to use our performance framework. In this project, our performance framework consisted of the following tooling:

- Maturity Assessment: Gobenchmark
- Load injection: Gatling and LoadRunner
- Script development: InteliJ
- CICD: Jenkins
- Version Control: Git
- Monitoring: Prometheus and Dynatrace
- Workload modeling: Performance Toolbox
- Reporting: Confluence
- Performance Board and Defect Tracking: Jira

### 4.2 Automation of Performance

Manual performance test execution and analysis is time-consuming and prone to human errors. Thanks to continuous integration solutions such as Jenkins and plugins from Dynatrace, we automated test executions and configured performance quality gates. As mentioned earlier, late detection of performance problems is expensive. By increasing the performance engineering maturity, we empowered this organization to capture the true value of performance. Figure 6 outlines the impact of the performance maturity level on organizations. When they improve their practices, they detect and solve performance problems earlier and reduce defect costs as mentioned in section 3.2.

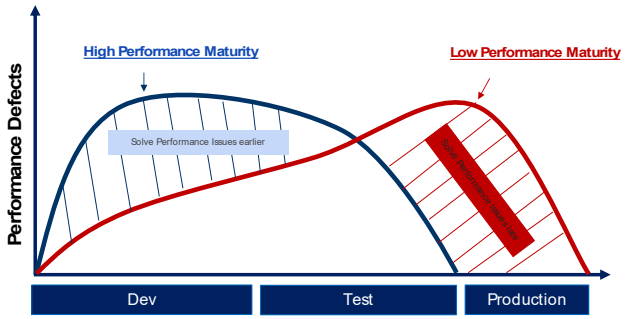



**Fig. 6.** A low-performance maturity level results in more performance problems detected in production and higher performance defect costs.

Looking back on this three-year project, I realize it was an incredible time. We established a culture of performance and prevented hundreds of performance issues from getting flawed to production.

### 4.3 Challenges in this Performance Engineering Project

We've created a culture of performance in this organization, which is the most important one from my perspective. If performance is in everyone's mind, the team will consider it during the entire software development process.

The five challenges below need to be solved from a technical standpoint.

1. Isolation of services and applications under test: Smaller test stages and reduced capacity on 3rd party services impacted performance test results.
2. Test data management: Performance tests generate huge data volumes. Assignment of test data sets and housekeeping is highly recommended.
3. Technical debts: The developers' workload is high, and they cannot include performance defects in their monthly sprints.
4. Workload models: New products result in difficult-to-predict transaction patterns. Workload modeling should be executed frequently in production to gain new insights for subsequent performance sprints.
5. Performance Engineering skills: To learn this science, more than introductory courses are required. Universities could integrate performance engineering lectures into their syllabus to educate the next generation of performance engineers.

# 5 PERFORMANCE ENGINEERING MATURITY

On the one hand, every organization could find its way to develop and operate reliable applications. On the other hand, we could share good practices and increase the chances that everyone would implement better business applications.

Based on our experience, one of the significant challenges is convincing leadership and creating awareness for performance engineering. To solve this business **problem,** we've invented [9] the performance engineering maturity knowledge model and implemented its algorithms in the Gobenchmark platform. At Gobenchmark, we combine human-AI-powered knowledge with qualitative analysis, making the unmeasurable measurable and bringing flexibility to changes in markets, customers, and technologies. Figure 7 outlines the core elements of Gobenchmark, which are:

- Advice from industry experts.
- Framework-based Analysis.
- Rating and comparison.
- Remediation.

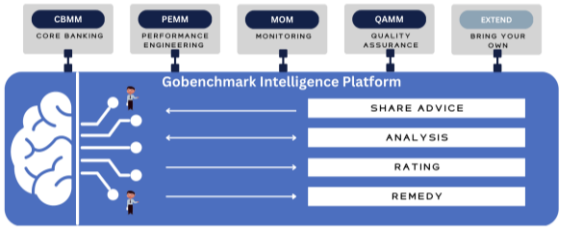



**Fig. 7.** The *Gobenchmark* platform, including its knowledge models and core features Advice, Analysis, Rating, *and Remedy.*

### 5.1 Advice from industry experts

Classic maturity models often fail because they do not adapt to changes. Collecting the latest information about methods and tools is crucial to avoid outdated knowledge. In Gobenchmark, we created a share advice catalog [6], allowing every industry professional to share good practices and hints about their solutions and how they rate their practices and tools. Furthermore, we store such advice in a flexible and reusable format to ensure that the built-in AI can utilize this information when creating recommendations for a customer's remediation plan.

### 5.2 Framework Analysis

At Gobenchmark, we have implemented framework-based analysis because it generates descriptive and explanatory conclusions. The interviewee walks through 27 questions structured in domains and practices. Each practice can be answered by choosing Always, Often, Rarely, or Never. After completing the assessment, Gobenchmark will calculate and present the performance engineering maturity score. Such ratings are easy to understand and allow a comparison to peers or industry standards. Gaps can be identified by showing how teams or organizations are rated, and remediation actions can be derived.

### 5.3 Rating and Comparison

We have no rating for the performance maturity of business applications or organizations. A high CMM level is no indicator of performant, secure, and well-designed IT services. Nevertheless, the outcome of our framework analysis in Gobenchmark can be transformed to a rating from C- to A and indicates how organizations or teams are adopting industry best practices.

By seeing the rating, we can identify blind spots, compare businesses to their peers, and create a remediation plan. In Gobenchmark, we show the rating immediately after the framework analysis, which creates essential benefits:

- We understand gaps much faster.
- We can focus our efforts on critical blind spots.
- We have everything we need to show a comparison to industry standards and peers.
- We can build the remediation plan based on identified gaps expressed by lower scores.

The performance engineering benchmark in our Gobenchmark platform is dynamic and will be re-calculated month by month.

### 5.4 Remediation

For performance engineering to work, it must take us on a journey where we learn concepts as we do things. Seeing gaps expressed by a rating does not solve these problems. If we leave organizations alone to solve these shortcomings, they might run into further issues, such as going in the wrong direction.

The AI-powered brain of Gobenchmark provides the expected guidance. It analyzes a customer's assessment results, incorporates knowledge from industry experts, and creates a remediation plan

that shows how organizations can reach the next level by improving their practices and methods and using better tools.

Our world is changing extremely fast, and we can't expect our current approach to work tomorrow or several weeks ahead. Knowledge from industry experts helps. However, we also see challenges in getting relevant insights from subject matter experts. For this reason, we've integrated large language models to acquire domain and practice-specific advice, which we incorporate into the AI-powered remediation plans to provide better customer mentorship.

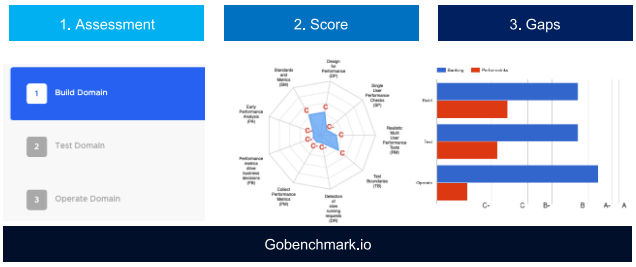



**Fig. 8.** In three steps from self-assessment through scoring and remediation.

## 6 OUTLOOK AND CONCLUSION

We believe that reliable business applications must become a commodity and should be achievable by every business. Today, the limiting factor [3] is mainly knowledge and awareness. Not knowing how to integrate performance or observability into value streams can set your business at risk. Organizations might spend too much time reinventing the wheel while their competitors adopt industry standards and dramatically reduce their development efforts.

To adopt performance engineering practices much faster and lower the risks involved, we propose the [9] "Performance Engineering Maturity Model." Figure 8 outlines how we use Gobenchmark in our performance engineering project. The three steps to improve the performance engineering maturity are:

1. Assessment to get guidance on integrating performance practices into organizations' value stream.
2. Scoring to raise awareness and highlight organizations' adoption of industry standards.
3. Remediation to solve technical and methodical gaps much faster and save time by avoiding reinventing the wheel.

**The benefits of using Gobenchmark are:**

- Safe time because we understand gaps and get a remediation plan within a few minutes.
- Reduce risks because we follow industry best practices.
- Simplify things because you get guidance along the way.
- Avoid DIY (Do it yourself) because Gobenchmark shares practice-proofed methodical and technical insights with us, so we no longer need to reinvent the wheel.
- Reduce costs because a higher maturity level helps our teams avoid expensive reliability issues in the first place.

Read more about the Gobenchmark platform on this page https://gobenchmark.io/.

## ACKNOWLEDGMENTS

I thank the performance engineering community for sharing their knowledge. Also, I am very grateful for the hard work of our Performetriks team on developing products such as Gobenchmark to make performance engineering scalable and protect thousands of businesses from learning performance the hard way.

## REFERENCES

[1] **Applied Software Measurement**, Capers Jones, 1996.
[2] Jim Collins. Good to Great. 2001
[3] Josef Mayrhofer. 2023. *Human-AI powered Strategies for Better Business Applications.* Performetriks, Minnesota. https://link.springer.com/chapter/10.1007/978-3-031-35734-3_26
[4] M. Woodside, G. Franks, D. C. Petriu. 2007. *The Future of Software Performance Engineering*. IEEE. https://ieeexplore.ieee.org/abstract/document/4221619
[5] Peter J. Pronovost, George J. Ata, Brent Carson, Zachary Gordon, Gabriel A. Smith, Leena Khaitan, and Matthew J. Kraay. 2022. *What Is a Center of Excellence.* Liebertpub. https://www.liebertpub.com/doi/abs/10.1089/pop.2021.0395?journalCode=pop
[6] Josef Mayrhofer: Performetriks.: https://www.performetriks.com/blog/categories/gobenchmark. (2023)
[7] José Raúl Capablanca,Good Reads Quotes, https://www.goodreads.com/quotes/650766-in-order-to-improve-your-game-you-must-study-the .
[8] E.W. Fleisch, Markus; Wortmann, Felix, Geschäftsmodelle im Internet der Dinge. HMD Praxis der Wirtschaftsinformatik. 51(6) (2014) 812-826.
[9] Josef Mayrhofer: Performetriks.: https://ppubs.uspto.gov/dirsearch-public/print/downloadPdf/11847597. (2023)

# Green Software Metrics

Andreas Brunnert
Munich University of Applied Sciences HM
Munich, Germany
brunnert@hm.edu

## ABSTRACT

Efficiency has always been at the core of software performance engineering research. Many aspects that have been addressed in performance engineering for decades are gaining popularity under the umbrella of Green IT and Green Software Engineering. Engineers and marketers in the industry are looking for ways to measure how green (in terms of carbon dioxide emissions) their software products are. Proxy measures are proposed, such as hosting cost or the power consumption of the hardware environment on which the software is running. In environments where a software system runs on a dedicated server instance, this may make sense, but in virtualised, containerised or serverless environments, it is necessary to find ways of allocating the energy consumption of the entire server to software components that share the same infrastructure. This paper proposes the use of resource demand measurements as a basis for measuring how green a given software actually is.

## CCS CONCEPTS

• **Software and its engineering → Software performance**.

## KEYWORDS

Green IT, Green Software Engineering, Resource Demand

**ACM Reference Format:**
Andreas Brunnert. 2024. Green Software Metrics. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3629527.3652883

## 1 INTRODUCTION

In order to meet global carbon dioxide (CO2) reduction targets, the IT industry needs to be able to quantify its emissions. Without quantifiable emissions, it is difficult to identify improvements and assess their impact. While it is common to use the energy consumption of servers or entire data centers to derive their CO2 emissions [1], there is no comparable metric for software [4]. Modern software systems run in virtualised, containerised or serverless infrastructures, for which we need to find ways of allocating the CO2 emissions of entire servers to individual software components or transactions [5].

To achieve this goal, several proxy approaches are currently being used in the industry [3, 4]. One proxy for carbon emissions is the hosting cost of a software system [3] (e.g., when renting instances from a cloud provider). The disadvantage of this approach is that the price of the runtime environment does not correlate exactly with the resource or CO2 emissions of the software. Just looking at the price differences for the same runtime environment on a cloud environment when choosing different payment options (e.g., up-front or on-demand) shows that there is no direct correlation between price and resource use.

Another proposal from the Green Software Foundation is the Software Carbon Intensity (SCI) specification[1]. SCI defines the carbon emissions of a software for a given unit of work (e.g., a request to the system). The SCI specification combines the carbon emissions of all components and transactions of a software system into a single rate value. The advantage of this approach is the simplicity of the result but it makes it difficult to understand the carbon emissions of specific transactions or components of a system.

As an alternative approach, the Green Software Measurement Model (GSMM) [4] attempts to describe a reference model for assessing the resource and energy efficiency of software products and components. GSMM focuses mainly on the individual components of a software system, without considering their interrelationships while processing individual units of work (e.g., transactions). Therefore, the authors [4] also note that the current GSMM methods are not fully applicable to complex architectures or distributed systems.

To overcome the limitations of the above approaches, this work proposes the use of resource demand measurements at the level of individual components and transactions as a basis for measuring how green a software is.

## 2 RESOURCE DEMAND MEASUREMENTS AS GREEN SOFTWARE METRICS

Measuring or calculating [6] the resource demands (i.e., CPU, memory, storage, network) of software systems is common in the software performance engineering community, as such data is required for capacity planning and performance modeling techniques. We propose to use the same data to quantify the emissions of a software system on a given runtime environment.

When measuring resource demand for a specific transaction, typical metrics collected are CPU time, bytes allocated in memory, or bytes written to/from storage or the network. Many of these metrics, such as the amount of memory consumed by a transaction or the amount of bytes written to or read from storage or the network, are independent of the underlying hardware, as they are primarily influenced by the parameters of a particular transaction. The only metric that is tied to the actual processor used during the measurements is CPU time.

In a previous work we have already shown and evaluated the ability to measure all these resource demands of a software system

[1] https://sci.greensoftware.foundation

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*ICPE '24 Companion, May 7–11, 2024, London, United Kingdom*
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652883

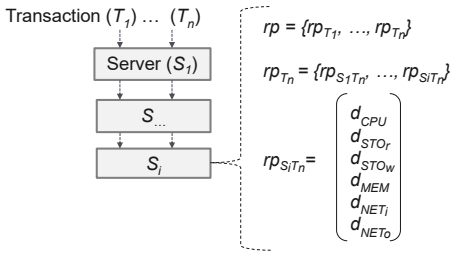



**Figure 1: Resource Profile [2]**

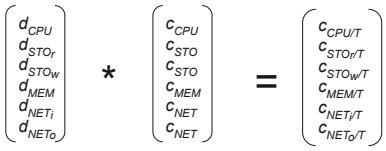



**Figure 2: Calculating Carbon Emissions per Transaction**

[2]. Therefore, we propose to use this data as a basis for assessing the carbon intensity of a software system. To structure the data we use so-called resource profiles, which can store the data for each transaction of a given software system separately by server (see Figure 1), software component or even at the level of individual operations [2]. A resource profile (*rp*) is a set of vectors (i.e., $rp_{T_n}$) that describe the resource demand (*d*) for individual transactions (*T*, numbered from *1* to *n*) for a specific workload and a certain set of servers (*S*, numbered from *1* to *i*). Resource profiles contain resource demands for the following resource types: CPU ($d_{CPU}$), storage (differentiated by read $d_{STO_r}$ and write $d_{STO_w}$ operations), memory ($d_{MEM}$), and network (differentiated by incoming $d_{NET_i}$ and outgoing $d_{NET_o}$ traffic).

A resource demand vector ($rp_{S_i\,T_n}$) of a transaction on a given server (virtual machine, container or serverless component) can now be used to derive carbon emission metrics based on the carbon intensity of the underlying hardware components as shown in Figure 2. For this calculation we need to be able to quantify the carbon emissions (*c*) of the different resource types (CPU: $c_{CPU}$,storage: $c_{STO}$, memory: $c_{MEM}$, network: $c_{NET}$) in the same unit as the resource demand is stored in the vector. For CPU, we need to know how much carbon is emitted when a core is running at a certain utilisation level; for memory, the carbon emissions for a given size (e.g., GB); and for storage and network, how much carbon is emitted when a given amount of data is processed. This data can be collected using services such as climatiq[2] for CPU, storage and memory for all common cloud environments, instance types and regions. Based on the measured resource demand data and carbon emissions of the individual resources the carbon emissions for individual transactions of a software system can be calculated as shown in Figure 2 by mutliplying the resource demand data with the carbon emissions. The use of both data sources allows the carbon intensity of software to be derived in more detail than is currently possible in industry.

To evaluate the feasibility of the aforementioned proposal, we are currently implementing a prototype based on the architecture

[2]https://www.climatiq.io

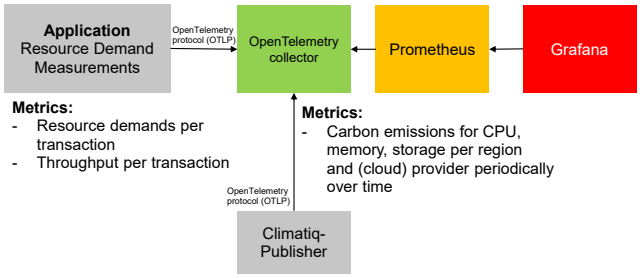



**Figure 3: Prototype Architecture**

in Figure 3. We are using OpenTelemetry[3] as a standard for transferring the required metrics from the application and climatiq to Prometheus[4] as a time series database. We need to store both sets of data (resource demand and emission data) correlated by time as emissions vary over time depending on the amount of green energy used to power the runtime environment (e.g., lack of solar power at night). Grafana[5] is used to visualize the results of this calculation to show the carbon emissions per transaction as well as per server, container, or virtual machine involved in processing a transaction over time.

## 3 CONCLUSION & FUTURE WORK

The use of resource demand measurements helps to provide a more detailed insight into how much carbon a software system emits (in other words, how green it is). We are currently building a prototype that links all the different parts together (resource demand data collection, carbon emission data collection and calculation). Once this work is done, we plan to run software experiments to evaluate our proposal against other approaches on the market, such as the SCI or the GSMM[4]. We also do not currently include the carbon emissions of network traffic in our prototype as we do not have the necessary emissions data, but we are looking for suitable data sources to fill this gap in the future.

## REFERENCES

[1] L. Belkhir and A. Elmeligi. Assessing ict global emissions footprint: Trends to 2040 & recommendations. *Journal of Cleaner Production*, 177:448–463, 2018.
[2] A. Brunnert and H. Krcmar. Continuous performance evaluation and capacity planning using resource profiles for enterprise applications. *Journal of Systems and Software*, 123:239–262, 2017.
[3] A. Currie, S. Hsu, and S. Bergman. *Building Green Software.* O'Reilly Media, Inc., 2024.
[4] A. Guldner, R. Bender, C. Calero, G. S. Fernando, M. Funke, J. Gröger, L. M. Hilty, J. Hörnschemeyer, G.-D. Hoffmann, D. Junger, T. Kennes, S. Kreten, P. Lago, F. Mai, I. Malavolta, J. Murach, K. Obergöker, B. Schmidt, A. Tarara, J. P. De Veaugh-Geiss, S. Weber, M. Westing, V. Wohlgemuth, and S. Naumann. Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—green software measurement model (gsmm). *Future Generation Computer Systems*, 155:402–418, 2024.
[5] A. Katal, S. Dahiya, and T. Choudhury. Energy efficiency in cloud computing data centers: a survey on software technologies. *Cluster Computing*, 26(3):1845–1875, June 2023.
[6] F. Willnecker, M. Dlugi, A. Brunnert, S. Spinner, S. Kounev, W. Gottesheim, and H. Krcmar. Comparing the accuracy of resource demand measurement and estimation techniques. In M. Beltrán, W. Knottenbelt, and J. Bradley, editors, *Computer Performance Engineering*, pages 115–129, Cham, 2015. Springer International Publishing.

[3]https://opentelemetry.io
[4]https://prometheus.io
[5]https://grafana.com

# Author Index