Fast, Light-weight, and Accurate Performance Evaluation using Representative Datacenter Behaviors

Jaewon Lee*
jaewon@meta.com
Meta
Menlo Park, CA, USA

Dongmoon Min
dongmoon.min@snu.ac.kr
Seoul National University
Seoul, Korea

Ilkwon Byun
ik.byun@snu.ac.kr
Seoul National University
Seoul, Korea

Hanhwi Jang†
hanhwi@ajou.ac.kr
Ajou University
Suwon, Korea

Jangwoo Kim
jangwoo@snu.ac.kr
Seoul National University
Seoul, Korea

ABSTRACT

Datacenters rapidly evolve by adopting new features such as new hardware deployment and software patches. Adopting a new feature requires an accurate evaluation of its impact to minimize the risk to the multi-million dollar computing infrastructure. However, a comprehensive performance analysis of a datacenter is extremely challenging due to its cost and multitenancy. Evaluating the performance in a live datacenter is accurate but prohibitive to prevent any damage to production services. Using conventional load-testing benchmarks on small-scale testbeds is imprecise as they do not consider the effect of other co-located jobs.

In this paper, we propose *FLARE*, a fast, lightweight, and accurate performance evaluation method using representative datacenter behaviors. The key idea is to extract a small set of representative job colocation scenarios from all possible job colocations in a target datacenter. FLARE systematically characterizes and groups job colocations according to performance and resource metrics, providing high-level insights into the datacenter's behaviors. Then, it reconstructs the colocations on a testbed and allows accurate feature evaluation with load-testing benchmarks. We evaluate FLARE using an in-house datacenter and three features: cache sizing, DVFS, and SMT configurations. FLARE accurately estimates the impact of features with less than 1% errors by incurring 50× and 10× lower evaluation costs compared to full datacenter and sampling-based evaluation, respectively.

CCS CONCEPTS

• **Computing methodologies** → **Simulation types and techniques**; • **Computer systems organization** → *Architectures*; • **Information systems** → *Data centers*.

*This work was done before the author joined Meta.
†A corresponding author

Middleware '23, December 11–15, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0177-1/23/12…$15.00
https://doi.org/10.1145/3590140.3629117

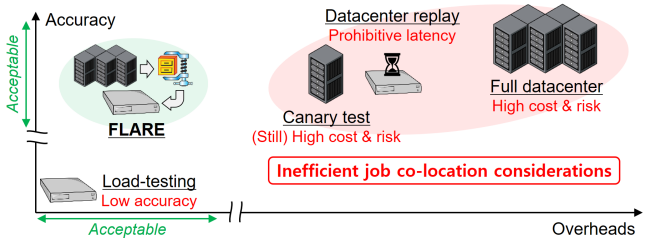


Figure 1: The accuracy and overheads of the existing datacenter performance evaluation methodologies.

KEYWORDS

datacenters, performance modeling, sampling-based evaluation

ACM Reference Format:
Jaewon Lee, Dongmoon Min, Ilkwon Byun, Hanhwi Jang, and Jangwoo Kim. 2023. Fast, Light-weight, and Accurate Performance Evaluation using Representative Datacenter Behaviors. In *24th International Middleware Conference (Middleware '23), December 11–15, 2023, Bologna, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3590140.3629117

1 INTRODUCTION

Datacenters advance by continuously adopting new *features* such as new hardware, software, and system configurations to improve performance and efficiency. Performance evaluation is the most critical step in this evolutionary process because it affects deployment decisions which can save or cost millions of dollars. For instance, inaccurate performance estimation of new hardware might lead to purchasing sub-optimal hardware in datacenter scale, causing irrecoverable financial damage. Datacenter engineers, therefore, do their best to accurately estimate the features' impacts before the adoption.

However, accurate and efficient performance evaluation of datacenters is extremely challenging. Figure 1 illustrates the limitations of previous datacenter evaluation methods. First, we conventionally use load-testing benchmarks on small-size testbeds to analyze the performance of datacenter workloads. The previous datacenter benchmarks [6, 28, 30, 77, 90, 96, 106] help to analyze and to estimate the feature's performance impact (e.g., tail latency) on a single datacenter job. However, the benchmarks do not consider the interference from other colocated jobs in a datacenter [15, 18],

leading to inaccurate performance evaluation in a datacenter where multiple jobs are co-scheduled.

Next, performance evaluation using live datacenters is accurate but prohibitive due to its *high costs and risks*. The overheads come from a datacenter's large performance variance [50, 95, 104]. To embrace such variance and obtain accurate results, a feature *must* be evaluated using many datacenter machines and jobs despite high deployment overheads and large-scale service disruption risks. A recent study [58] proposes a statistical approach to construct a small canary cluster and reduce live evaluation costs; however, it still suffers from nontrivial overheads (tens to hundreds of machines) and the possibility of damaging production jobs.

In this paper, we propose *FLARE*, a fast, lightweight, and accurate performance evaluation method using representative datacenter behaviors. The main idea is to reconstruct the job colocation behaviors of a datacenter onto a small-size testbed, allowing accurate performance evaluation with conventional load-testing benchmarks. However, as there are an enormous number of job colocations from tens of thousands of machines and jobs in a datacenter, it is not affordable to naively reconstruct all job colocations.

To address this challenge, we introduce a systematic way to characterize and extract representative datacenter job colocations and a performance evaluation method using those representatives. Our insight is that a datacenter's machine behaviors and job colocations would have similarities and redundancies, and we should be able to classify them into a small number of groups which well represent the overall datacenter characteristics.

To effectively obtain the representative behaviors, we first collect and analyze a large number of performance and resource metrics (e.g., IPC, cache miss, CPU utilization) from datacenter jobs and machines, and translate them into a smaller number of *high-level metrics* (e.g., metric indicating CPU intensive + frontend bandwidth bound + ALU-intensive behavior). This not only reduces the data dimension for easier grouping but also identifies the key constituents of datacenter performance. In our environment, we translate 100+ raw performance/resource metrics into 18 high-level explanations.

Based on the lessons from these metrics, we then classify the datacenter behaviors into a small number of groups and select the representatives. By reproducing the *job co-location scenarios* of these representatives (i.e., representative scenarios) with load-testing benchmarks, we can 1) quickly and accurately estimate the comprehensive performance impact of a feature on a datacenter without reproducing the whole datacenter behaviors and 2) easily reason about why a feature makes such an impact, thanks to our high-level description of each representative scenario.

We emphasize that FLARE is a *generic* methodology which can be applied to an arbitrary datacenter environment adopting an arbitrary feature (e.g., SW/HW upgrades, configuration updates). For our in-house datacenter consisting of 895 co-location scenarios, FLARE accurately extracts 18 representatives and accurately evaluates three different features – cache sizing [71], DVFS configuration, and SMT configuration [94] – with 50× lower overheads and minimal errors (~1%). We also show that sampling-based evaluation and conventional load-testing benchmarks suffer from high overheads and errors.

In summary, the contributions of this paper are as follows:

- **Problem identification.** We demonstrate that the current datacenter feature evaluation practices are either too expensive or inaccurate. We also identify the fundamental reason – the lack of efficient job co-location reproducing idea – to develop our solution.
- **Simple and effective solution.** We propose a generic and easy-to-apply solution to address the issue. FLARE can summarize an arbitrary datacenter environment and deliver accurate feature evaluation results and insights using representative scenarios.
- **Strong validation results.** We successfully reduce the datacenter evaluation overhead by 50× without losing the accuracy. This is much more efficient than sampling-based evaluation (~10× overhead reduction), and much more accurate compared to conventional load-testing benchmarks.

The rest of this paper is organized as follows. Section 2 introduces the existing performance evaluation methodologies and their limitations. Section 3 describes the challenges of fast and accurate performance evaluation with a case study from our datacenter. We discuss the details of FLARE in Section 4. Section 5 shows the effectiveness of our method with evaluation results. We discuss related work in Section 6 and conclude the paper in Section 7.

2 BACKGROUND AND MOTIVATION

In this section, we first introduce recent datacenter-improving studies and their performance evaluation methods to discuss the need for FLARE.

Datacenters, or warehouse-scale computers [40, 75], have become the standard platform for hosting modern internet services. The distinguishing trait of this platform is that they host *multiple* scale-out jobs on *shared resource* consisting of thousands of machines [87, 95]. A large number of studies aim to improve the performance and efficiency of such a complex computing environment.

Improvements from features. A group of studies improve datacenter performance by introducing *features* (i.e., new software, hardware, configurations, or operation policies) to each machine. The examples include better resource isolation capabilities [60, 92, 93, 99, 104], smarter power management policies [43, 44, 52, 66, 67, 76], and novel architectural improvements [11, 12, 37]. In fact, any generic architecture or system improvement (e.g., better prefetchers, optimized libraries) can be considered a feature to improve datacenters.

Feature evaluation challenges. Table 1 summarizes the evaluation setups of the studies mentioned above. First, we notice that some studies simply validate their idea with conventional load-testing benchmarks with individual applications [6, 11, 28, 37, 38, 43, 44, 52, 59, 66, 76–78, 92, 108]. While this practice provides insights regarding a feature's impact on a specific job, we *cannot* naively expect the same impact for much more complex job-colocation scenarios of datacenters. In Section 3, we show examples where load-testing benchmarks deliver highly inaccurate estimations.

Second, we notice that many other studies use either actual datacenters or environments of similar scale for feature evaluation [16, 19, 20, 22, 33, 49, 50, 58, 64, 65, 95, 99]. This approach incurs prohibitively high costs, risks, and latencies, and eventually slows

Evaluation scale	Performance evaluation methodology			
	Individual application	Handcrafted co-location	Scheduler-based co-location	Datacenter-scale co-location
Single machine	Memory Blades [59] Kanev et al. [52] Smoothoperator [43] Adrenaline [44] Power Routing [76] PowerNap [66] DjiNN and Tonic [37] CloudSuite [28] Perfkit Benchmarker [6] TPC benchmarks [77]	Prophet [11] Dirigent [108] Sirius [38] Tang et al. [92] Octopus-man [78]		
Small evaluation cluster (≈10∼100 machines)	BigDataBench [96] SPECvirt [15]	DeathStarBench [30] Heracles [60] CPI2 [104]		
Datacenter scale (≈100∼1000 machines)		Bubble-Up [65] Bubble-Flux [99] HCloud [21]	Paragon [19] Quasar [20] Carbyne [32] Graphene [33] Morpheus [49] Zhang et al. [107] Baymax [12] Tarcil [22] Whare-map [64]	Resource Central [16] Kambadur et al. [50] Borg [95] Tang et al. [93] Kanev et al. [51] Ren et al.[83] Hazelwood et al. [39] TPU [48] Ayers et al. [3] Delimitrou et al. [23] WSMeter [58]

*Trace analysis and simulation approaches are classified as a single machine-individual application.

Table 1: Performance evaluation setups of state-of-the-art datacenter studies.

down assessing and deploying valuable new features which can significantly boost datacenter performance and efficiency.

To enable fast evaluation and adoption of features, FLARE aims to reduce the feature performance evaluation overheads by utilizing representative datacenter behaviors.

We note that some features drastically change the machine's shape (e.g., core count, RAM capacity) and hence the datacenter's job co-location landscape. In this case, the representative scenarios that we extract from the current datacenter would become less valid after the feature deployment, as the new datacenter will have quite different job co-location behaviors. Therefore, we limit our scope to the features which do not change the datacenter machine's shape; Section 5.5 discusses how we may handle such cases. We emphasize that the features without machine shape changes (e.g., software upgrades) are much more frequent and FLARE is crucial for successful datacenter management.

Improvements from datacenter schedulers. Another group of studies improve datacenter performance using new scheduling algorithms. Specifically, they control job co-location scenarios and global resource allocations [19–22, 64, 65, 78] to identify and avoid undesirable scenarios such as high resource contention and resource underutilization.

Scheduler evaluation challenges. Since realistic datacenter behaviors are necessary to measure the holistic impact of schedulers, these studies tend to use large-scale job co-location setups which incur high evaluation overheads. However, as these studies gain benefits by *altering* job co-location scenarios, it is difficult to define and utilize a single set of representative behaviors (i.e., job co-location scenarios) to reduce the performance evaluation overheads. Nonetheless, in Section 5.6, we discuss how FLARE can help optimize the evaluation of datacenter schedulers. We also emphasize that performance improvements from the features are much more frequent (e.g., periodic software upgrades) and FLARE can drastically improve the productivity of datacenter evolution process.

3 CHALLENGES OF DATACENTER PERFORMANCE EVALUATION

In this section, we demonstrate the challenges of accurate datacenter performance evaluation with a case study from our in-house environment. In this case study, we evaluate the effect of different cache size on our datacenter and show that conventional load-testing benchmarks *fail* to accurately estimate the feature's performance impact, which emphasizes the need for *accurate* datacenter performance evaluation method.

3.1 Pitfalls of (co-location unaware) conventional load-testing benchmarks

In this case study, we examine how the cache size affects our datacenter (Feature 1 in Table 4) and demonstrate that the performance impact measured from load-testing benchmarks alone does not necessarily translate into in-datacenter performance.

Datacenter setup. We design our datacenter following the prior work [58]. Our datacenter consists of three racks of homogeneous machines, and they host *High Priority (HP)* and *Low Priority (LP)* jobs listed in Table 3. HP job performance is our primary concern, and we ignore LP job performance as they run using free quota. Each job is containerized to ease the deployment in a scale-out manner. Please refer to Section 5.1 for more details.

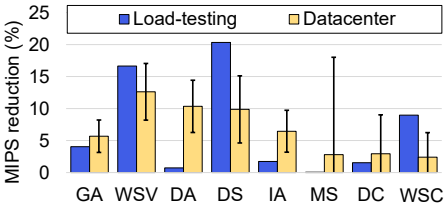


Figure 2: Evaluating the performance impact of cache sizes (Feature 1) with conventional load-testing benchmarks; MIPS reduction on the jobs in absolute value

Load-testing benchmark setup. We measure the feature's impact on each HP service with load-testing benchmark. Similar to previous works [51, 58], we populate instances[1] of each service on a single machine and measure the feature's impact on it.

Load-testing benchmarks fail to estimate the in-datacenter performance impacts. Figure 2 shows the performance estimations from the load-testing benchmarks and the actual impacts observed from datacenter machines. We use *Million Instructions per Second* (MIPS) as a performance metric of interest as our jobs are optimized to spend time in spin locks minimally, exposing their throughput as MIPS [58]. The datacenter results are the average of all instances of each service; error bars denote the standard deviation. The estimation of the benchmarks deviates from the actual result in our datacenter because the benchmarks do not take into account any interference from other co-located jobs.

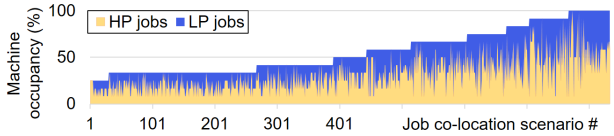
3.2 Challenges of realistically reproducing datacenter behaviors

To improve from conventional load-testing benchmarks, we need to reconstruct production datacenter behaviors or job colocations. Then what are the fundamental challenges that prohibit researchers from achieving this goal?

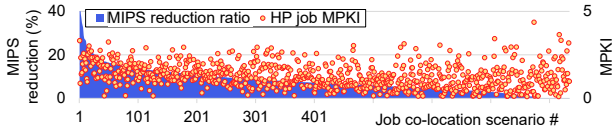
First, the most straightforward approach is to record every behavior of a datacenter and exactly reproduce it with benchmarks. However, this is practically infeasible considering the scale and diversity of the behaviors. Exact reproduction would require either a large number of machines to finish fast or extreme tolerance to the evaluation latency. Both are not viable options for datacenter feature assessment because we cannot afford a large number of machines for testing an experimental feature (especially if the feature requires purchasing new hardware), nor can we tolerate the latency to reproduce thousands of job colocation behaviors using few machines. Even if we use sampling [58] to reduce the number of machines to consider, it still requires a large number of machines.

Second, we may extract and utilize representative datacenter behaviors. However, datacenter environments are difficult to analyze due to their complexity (i.e., job colocation). Figure 3a shows the machine occupancy characteristics of our datacenter. The datacenter has 895 job colocation scenarios with a wide variety of HP/LP job mixes and utilization, similar to the observations from the other datacenters [81, 82, 97]. As a result, the scenarios have

[1]Instances of a job are identical processes/binaries which run in a distributed manner to share the loads.



(a) Machine occupancy characteristics. The scenarios are sorted by total machine occupancy. The occupancy shows step-like pattern as our jobs run as containers of a fixed size.



(b) Performance impact (MIPS reduction) and the LLC MPKI of the HP jobs. The scenarios are sorted by performance impact

Figure 3: Investigating the performance impact (absolute MIPS) of Feature 1 on our datacenter environment

vastly different resource demands, and we can expect each scenario to react differently to a feature.

Due to such complexity of job colocations, we cannot characterize them with simple metrics. Figure 3b shows the performance impacts of Feature 1 (i.e., cache sizing) on individual job colocation scenarios, along with MPKI, a highly relevant metric to cache size (Feature 1). Intuitively, Feature 1 should have a bigger impact on the scenarios with higher MPKI as it is related to memory systems. Under this assumption, we may heuristically select the scenarios covering various MPKI ranges to evaluate the feature. Unfortunately, the impact is not correlated to MPKI or any other single memory system-related resource metrics, as many other factors contribute to a job colocation scenario's performance. As a result, we cannot simply find representative datacenter behavior or job colocations based on a few highly relevant metrics, indicating the need for *systematic* way to analyze and extract representative datacenter behaviors.

4 FLARE: DATACENTER EVALUATION WITH REPRESENTATIVE SCENARIOS

In this section, we describe the details of FLARE. Figure 4 illustrates our framework and key operations. FLARE consists of `Profiler`, `Analyzer`, and `Replayer`, and performs the following four steps 1) data collection and refinement, 2) high-level metric construction, 3) clustering & representative behavior extraction, and 4) performance estimation with representative behaviors.

The following sections describe each step and its implementation in details.

4.1 Definition of datacenter behavior

Before introducing our methodology, we first present the precise definition of *datacenter behavior* (or *job co-location scenario*; we use these terms interchangeably throughout the paper) as it is our basic unit for performance evaluation.

Figure 5 illustrates how we define the scenario. Our datacenter consists of homogeneous machines, and each machine hosts jobs with different durations and resource demands (Section 5.1). As shown in the example, *every new combination of jobs* defines a

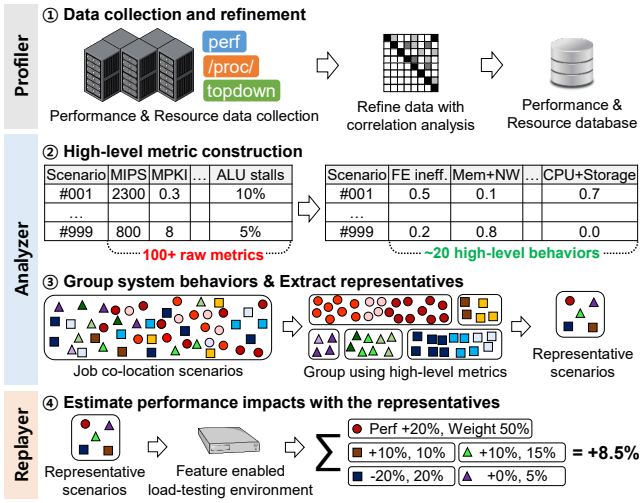


Figure 4: Overview of FLARE framework

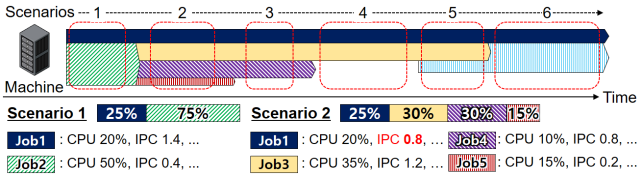


Figure 5: Definition of job co-location scenario

new scenario and we collect the scenarios from every datacenter machine. For each job in each scenario, we log the *average* performance and resource metrics to minimize the monitoring overheads. Note that a user may log extra information to take temporal/phase behaviors into account. For example, we currently log the average IPC of 1.4 for Job 1 in Scenario 1, but one may include standard deviations (e.g., IPC: 1.4±0.5) to enrich the temporal information.

4.2 Data collection and refinement

The first step of FLARE is to collect and refine performance and resource metrics from a datacenter.

Collection. We gather various performance and resource metrics from both software monitors (e.g., `/proc/` filesystem) and hardware performance counters (e.g., `perf`, Intel's `topdown` [100]) by implementing a deamon process, `Profiler`. It is deployed to all servers and periodically gathers system and microarchitectural statistics by using `perf, stats`, and `/proc/` filesystem. The collected statistics along with the commands and configurations of running jobs are recorded in our relational database for further analysis.

Figure 6 shows a subset of the metrics that we collect. We collect the metrics in two level – machine level (i.e., sum of the metrics from all jobs) and individual job level. As we only manage the performance of High Priority (HP) jobs and neglect Low Priority (LP) jobs running on free quota, we eventually have two versions of performance metric (e.g., LLC-APKI-Machine and LLC-APKI-HP). This two-level collection allows us to accurately model co-location

Metric	Description
L1-APKI	L1 accesses per kilo instructions
LLC-APKI	LLC access per kilo instructions
LLC-MPKI	LLC miss per kilo instructions
LLC-missrate	LLC misprediction rate
Workingset	Number of unique pages accessed per second
NIC read / write	NIC bandwidth (Mb/s)
I/O read / write	Disk bandwidth (MB/s)
CPU usage	CPU utilization
BPKI	Branches per kilo instructions
IPC	Instructions per cycle
Branch missrate	Branch misprediction rate
Cycles	Number of elapsed cycles under DFVS effect
Retiring	Nominal retiring instructions (% of the pipeline slot)
Bad Speculation (Branch,Flush)	Bad speculation events (% of the pipeline slot)
Frontend Bound (Latency, Bandwidth)	Frontend bound events (% of the pipeline slot)
Backend Bound (Compute, Memory)	Backend bound events (% of the pipeline slot)
Stores, Loads, ITLB, DTLB, Arithmetic, All/Far branches, x87, etc.	Number of instructions (% of total instructions)

Figure 6: The performance and resource metrics collected in our datacenter environment.

behaviors; we monitor the jobs of interest (*-HP) as well as their running environment (*-Machine).

Refinement. We notice that many of the metrics we collect are in fact duplicates. For example, we find that memory bandwidth reports from our monitoring tool is in fact a simple multiplication of LLC miss count and the payload size. Eliminating such highly correlated metrics allows us to reduce 100+ metrics to 85 metrics with weaker correlations. We point to the other studies [51, 69, 87, 95, 101] for collecting the metrics embracing such noises of datacenter environments.

4.3 High-level metric construction

As the second step, `Analyzer` constructs high-level metrics from the collected statistics, which will be the new and more insightful descriptors of the job co-location scenarios (Figure 4-②).

To reduce the dimensionality of the data, we first normalize each metric to have zero mean and unit variance, eliminating the biases from the metrics' inherent magnitudes. We then apply Principal Component Analysis (PCA) to represent the information in a lower dimensional space. We chose this method as it has proved effective for many architectural studies [24, 25, 47, 74, 80] as well as provides high-level metrics with better interpretability compared to the other non-linear techniques [102, 103]. Specifically, PCA reduces the dimension of the data using the linear combination of the original dataset's basic vectors. Therefore, it is easy to understand how the raw metrics contribute to the construction of the Principal Components (PC, i.e., high-level metrics). As each PC explains certain amount of variance in the original dataset (by design), we can also identify which PCs have more classification power. Lastly, the PCs are orthogonal to each other; we do not have to worry about one high-level metric moving along with the others when analyzing the results. For our environment, we obtain 18 PCs as illustrated in Figure 7.

While the general dimensionality reduction idea is quite similar to existing studies, FLARE has few major differences. First, many studies extract the PCs, identify the contribution of the original metrics (e.g., LLC miss is the major contributor of the first PC), and perform analysis in the *original metric's space* instead of the PC's space. For our environment, we found it nearly impossible to do so because there are too many original metrics of high importance.

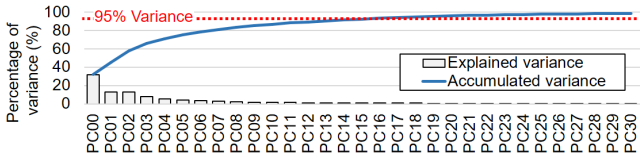


Figure 7: Determining the optimal number of Principal Components (PC). We select 18 PCs to explain 95% of the variance of the original dataset.

Also, the analysis targets of previous studies require only few PCs to describe their nature, allowing easy and straightforward analysis. Our datacenter behaviors are far more complex and we cannot simply investigate few PCs to get insights.

Therefore, we decide to interpret and label each PC to attribute a higher-level meaning to it. Figure 8 shows the PCs extracted from our datacenter along with their interpretations. We note that both the machine and the HP job metrics play an important role in forming the PCs.

Thanks to our two-level metric collection (Section 4.2), we observe some interesting traits unique to co-location environments. For example, PC10 indicates memory bottleneck in HP jobs combined with non backend-bound (hence non memory-bound) machine behavior. This implies that the HP jobs occupy a small portion of the whole machine and thus have a low influence on the overall machine behavior. The other metrics also depict very interesting and surprisingly specific combination of HP job and machine characteristics.

In summary, we successfully translate a large number of performance and resource metrics into valuable high-level insights, which are specialized to describe job co-location scenarios.

4.4 Representative behavior extraction

The third step is to group similar job co-location behaviors using the high-level metrics, so that we can identify a small number of representative scenarios; in Figure 4-③, similar shapes and colors indicate similar scenarios. For simplicity, we implement the second and third step in `Analyzer` together.

We first normalize all the selected PCs to have zero mean and unit variance (i.e., perform whitening operation) to make each PC retain the same amount of information. Next, we perform data-driven clustering to group the scenarios. The major challenge of our case is that there are no correct cluster labels for each scenario (i.e., unsupervised clustering). It is therefore impossible to utilize advanced techniques such as cross-validation to easily reduce the bias. Instead, we utilize metrics such as Sum of Squared Errors (SSE) [85] and Silhouette Score (SS) [86] to evaluate the quality of each setup. SSE indicates the sum of the distance between each point and its cluster centers, and SS scores each point by comparing how similar it is to its cluster (vs. the others), and returns the average score over all points. We select 18 clusters based on the data shown in Figure 9. Note that we use K-means clustering to achieve good results, but alternatives (e.g., hierarchical clustering of [74, 80]) can also be applied.

As a result, we successfully group 895 different scenarios of our datacenter into only 18 groups. For each group, we extract a representative scenario by selecting the scenario nearest to the group's centroid (or average) behavior. In Section 5.2, we show that each representative has distinct characteristics which allow us to accurately estimate the overall performance.

4.5 Feature performance estimation using the representative scenarios

The last step is to estimate a feature's performance impact using the representative scenarios with `Replayer`. We first apply the target feature to our performance evaluation environment and use `Replayer` to reproduce the representative scenarios with load-testing benchmarks. It reconstructs the environment by executing the jobs with the recorded commands and options.

Each scenario will report different performance results as they have different resource and performance characteristics. To generate the single number summary of the impact, we weight the impacts of the representatives by the group sizes and get the average (Figure 4-④). The reason behind this weighting is that we are more likely to observe a scenario from a larger group. In Section 5.3, we show that this summary accurately estimates the features' holistic impacts.

5 EVALUATION

We now provide the results showing the effectiveness of FLARE and also share the insights from our experiments.

5.1 Experimental setup

Datacenter setup. We carefully reproduce the machine setups, job characteristics, and user behaviors of a real-world datacenter based on the design of our industry partner [58]. Our environment consists of three racks of homogeneous machines. We show how we handle heterogeneous machine configuration in Section 5.5. The details of machine specifications are provided in Table 2. Each rack has eight machines, and we dedicate one rack for reproducing the datacenter behavior and two racks for modeling the clients. One of the client machines acts as the job submission system which launches the jobs using Docker containers [70]. We ensure that the client and scheduler do not become the bottleneck of the job execution.

Job submission system. The simulated users submit HP and LP jobs to datacenter machines as containers. This reproduces the resource isolation abilities of typical datacenters. To model the scale-out nature of datacenter jobs, each container (i.e., job instance) consumes only up to 4 vCPUs. The user requesting more computing power must launch multiple instances (i.e., copies) of a job. This accurately reflects the resource management policies of our target datacenter.

Machine Setup	
CPU	Intel Xeon E5-2650 v4 (2 sockets, 24 vCPUs per socket)
DRAM	256GB DDR4 2400MHz, Four channels per socket
Disk	Intel 730 Series SSD (SATA 6Gb/s)
Network	Intel X710 10Gbps Ethernet

Table 2: Datacenter machine specifications.

PC	Major contributors		High-level interpretation
	Machine level	Job level	
00	(+) CPU usage, L1 access, LLC access, **Frontend/Bandwidth**, IPC (-) **Backend/Core**, BPKI	(+) CPU usage, L1 access, LLC access, **Frontend/Bandwidth**, IPC, **Arithmetic**, **DTLB**, ITLB, Branches (-) **Backend/Core**, BPKI	Frontend bandwidth bound with many arithmetic ops from HP jobs.
01	(+) LLC missrate, BadSpec/Flush, CPU usage (-) **Frontend/Latency**, Backend/Core	(+) LLC missrate, BadSpec/Flush, CPU usage (-) **Frontend/Latency**, Backend/Core, **LLC loads**, **Mem BW**, **ITLB BW**	Large LLC contention due to LP jobs
02	(+) **LLC access**, **LLC miss** (-) Retiring	(+) **LLC access**, **LLC miss**, **Backend/Mem**, **Workingset**, (-) Retiring	Memory intensive backend bound
03	(+) (-)	(+) **BadSpec/Branch**, **Frontend** (-) Workingset	HP jobs suffering from branch misprediction
04	(+) (-) **LLC stores**, LLC store misses	(+) **Backend/Core**, **NIC Write**, **Stores** (-) **LLC stores**, LLC store misses, Workingset	Large stores fitting in LLC. HP job's intensive NIC write
05	(+) **I/O Read**, **I/O Write** (-) **NIC read**, **NIC write**, LLC MPKI	(+) **I/O Read**, **I/O Write**, x87 instructions (-) **NIC read**, **NIC write**, LLC MPKI, Far branches	High disk activity with low NIC activity
06	(+) **NIC write**, **Branch misses** (-) **I/O write**, L1 APKI, CPU usage, ITLB	(+) **NIC write**, **Branch misses** (-) **I/O write**, L1 APKI, CPU usage, ITLB	HP job's intensive network writes with large branch misses
07	(+) Workingset, LLC APKI (-) BadSpec/Flush, **LLC MPKI**, BPKI	(+) **L1 APKI**, **L1 stores** (-) BadSpec/Flush, **LLC MPKI**, BPKI	Memory intensive HP jobs, but fitting in LLC
08	(+) **NIC read**, **NIC write**, **I/O Read** (-)	(+) **I/O Read**, Branch miss/Job (-) IPC	Disk read and network intensive machine
09	(+) Far branches, LLC missrate, **Frontend/Bandwidth**, **BPKI** (-) **Cycles**, **Branch missrate**, LLC APKI	(+) Far branches, LLC missrate, **Frontend/Bandwidth**, **BPKI** (-) **Cycles**, **Branch missrate**, LLC APKI	Large number of branches and high branch hit rate causing frontend bandwidth bottleneck. Low cycles indicate low activity
10	(+) Branches (-) **Backend**	(+) **Backend/Mem** (-) Frontend, Retiring	Memory intensive HP jobs combined with branch intensive LP jobs
11	(+) Far branches, **I/O read** (-) **NIC read**, **NIC write**, **I/O write**	(+) Far branches, **I/O read** (-) **NIC read**, **NIC write**, **I/O write**	Disk read intensive without network
12	(+) **I/O read**, Branch miss (-) **NIC read**, **NIC write**	(+) **I/O read**, **LLC MPKI**, Backend/Core (-) **NIC read**, **NIC write**	Disk read and memory intensive HP job without network, along with machine having many branch misses
13	(+) (-) BPKI, **NIC read**, **NIC write**	(+) **I/O read**, **NIC read**, **NIC write**, Backend/Mem (-) BPKI	Disk read and network intensive HP jobs, but the total occupancy in per machine is low
14	(+) **L1 APKI**, **LLC store**, **BPKI** (-)	(+) (-) **L1 APKI**	Non memory-intensive HP jobs with memory (store) and branch intensive machine / LP jobs
15	(+) Workingset (-) x87	(+) **I/O read**, **BPKI**, **Frontend/Bandwidth** (-) L1 APKI	Disk read and branch intensive HP jobs with small working set, combined with large workingset LP jobs
16	(+) (-) I/O read	(+) **Frontend/Latency**, **Backend/Core**, I/O read, I/O write (-) Retiring	Frontend latency and backend compute bound HP jobs with I/O activities
17	(+) **NIC read & write** (-)	(+) I/O read, Frontend/Bandwidth (-) **NIC read**, **NIC write**	HP jobs are not network intensive but the overall machine is network intensive

Figure 8: High-level metrics (principal components) of our datacenter. The plus sign (+) indicates that a raw metric has a positive weight on a PC, and the minus sign (-) indicates that the metric has a negative weight. We omit the metrics with small weights. For each PC, we highlight the major contributing raw metrics which guide us to interpret the meanings.

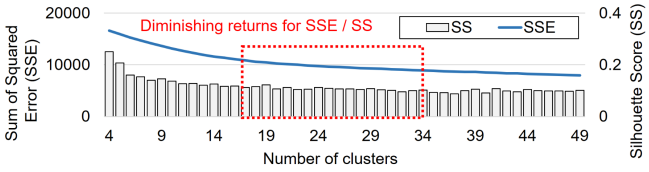


Figure 9: Investigating Sum of Squared Error (SSE) and Silhouette Score (SS) for various cluster counts. Lower SSE and higher SS indicates better clustering quality. The general guideline is to pick a point where the return (i.e., clustering quality) starts to diminish. We choose 18 clusters to strike the balance between the quality and cost.

The scheduler greedily runs a job in the datacenter machine with the least resource utilization for load-balancing purposes. As we do not overcommit the resources, saturation of the machines would result in a denial of scheduling requests. The length of each job is randomly determined at the submission time. Each job runs for at least 30 minutes to produce stable behaviors; server jobs (e.g., memcached) handle multiple user requests during its uptime, and the other jobs simply run the designated task (e.g., mapreduce [17]) for the given durations. The variation in the jobs' length and the users' request rates produce diverse resource behaviors (e.g., underutilization or saturation of machines).

Job characteristics. We use CloudSuite benchmarks [28] and SPEC CPU2006 [14] to reproduce datacenter job behaviors. Table 3 shows the details of the job configurations. Note that the metrics are the allocations per an instance (i.e., a 4 vCPU container). For HP jobs, we tune the working sets so that they match those of the our target datacenter. For LP jobs, as we observe both compute- and memory-intensive ones from the production environment, we select a subset of benchmarks from SPEC CPU2006 to reproduce similar behaviors.

We note that the selection of benchmarks may change depending on a datacenter's characteristics. We emphasize that FLARE does

High Priority (HP) jobs	
Data Analytics (DA)	Apache Hadoop with Mahout 4 maps, 4 reduces, run TrainNB phase 1 vCPU & 4GB DRAM per mapper/meducer
Data Caching (DC)	memcached 4 threads, 4GB working set, target QPS 100K
Data Serving (DS)	Apache Cassandra 20 threads, 16GB DRAM
Graph Analytics (GA)	Apache Spark 4 vCPU & 4GB DRAM for executor
In-memory Analytics (IA)	Apache Spark 4 vCPU & 4GB DRAM for executor
Media Streaming (MS)	Nginx 4 threads, 50 connections, dataset scaled
Web Search (WSC)	Apache Solr 12GB DRAM. Tomcat manages # threads
Web Serving (WSV)	MySQL, memcached, Nginx, PHP Default mySQL, Nginx settings with 2GB memory 2 threads & 2GB DRAM for memcached 5 threads for PHP
Low Priority (LP) jobs	
400.perlbench, 458.sjeng, 462.libquantum, 483.xalancbmk, 471.omnetpp, 429.mcf Four copies run in a container to consume 4 vCPUs.	

Table 3: Configurations of datacenter job instances.

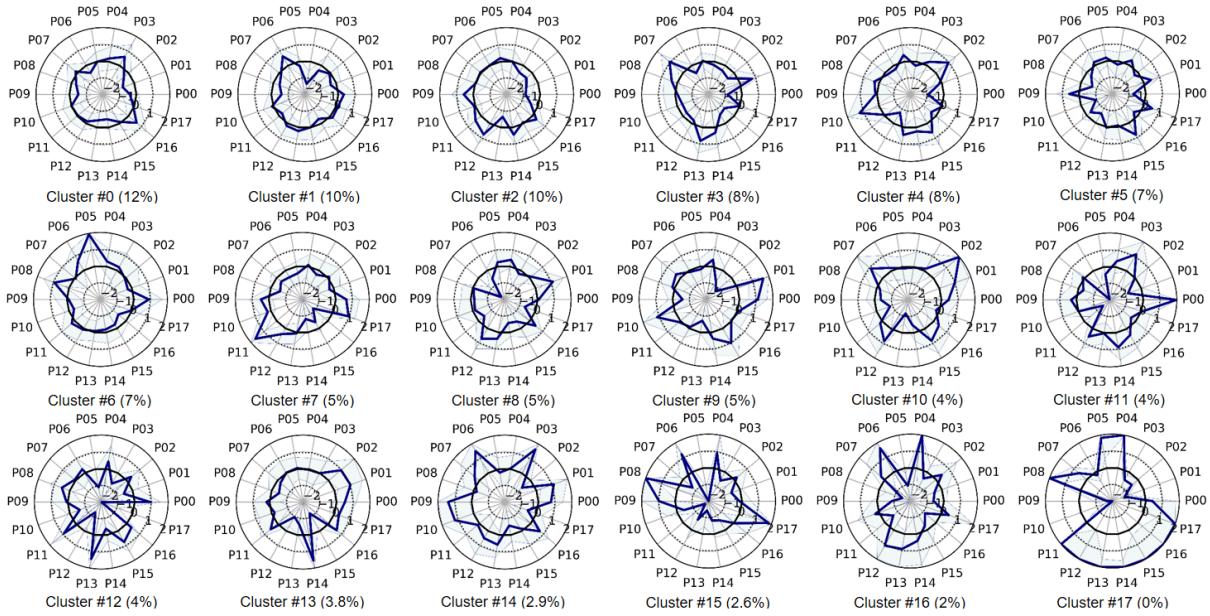


Figure 10: Clustering the datacenter system behaviors into a small number of groups. The radar plots show each group center's 18 PC values. The shaded regions indicate the ±1 standard deviation of the group. The dotted lines show ±1 standard deviation of the whole dataset. For each cluster, we also provide the weight in %.

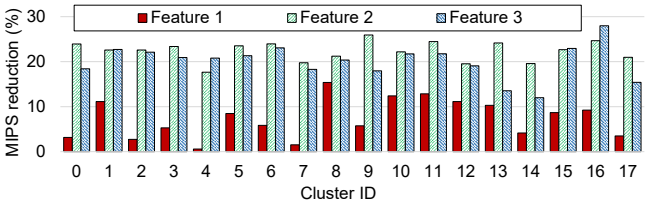


Figure 11: Performance impact (MIPS reduction) estimation from each representative scenario.

not limit benchmark or job types. Ideally, if we can thoroughly characterize the performance and resource behaviors of every job in the datacenter, we may utilize high-precision load generators such as iBench [18] to accurately reproduce the job behaviors.

Defining the performance. First, when calculating the performance, we only consider those of the HP jobs. LPBatch jobs run on free quota and we do not manage their productivity.

Since we have multiple different HP jobs, we define a summarizing performance metric to allow easier aggregation of per-job performance. Specifically, we use an instruction throughput-based performance metric similar to prior work [58], which is defined as follows:

$$Performance = Job\ MIPS\ /\ Job's\ Inherent\ MIPS$$

A job's inherent MIPS is the MIPS measured when the job runs alone on an empty machine. This allows us to prevent the jobs with inherently high MIPS from having higher importance. We chose this parameter mainly because our industry partner uses a similar metric, and many other studies show high correlation between the application-level throughput and the instruction throughput [57, 92, 104]. We emphasize that FLARE is not bound to any specific performance metric. Many alternatives [27] can be utilized, and the users with better definitions may use their own terminology.

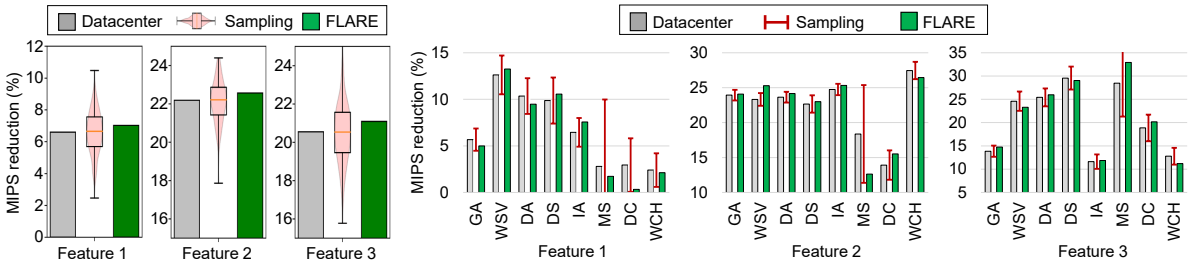
Datacenter improving features. We consider three different features to introduce performance changes. Table 4 shows the summary of the three features – cache sizing (Feature 1), DVFS policy (Feature 2), and SMT (Feature 3) – along with the baseline setup. For easier evaluation, we manipulate resource constraints via system setups to emulate hardware changes (Feature 1) and/or utilize the features which may *reduce* the machine's capability (Feature 2 and 3). It is therefore natural to observer performance degradation upon applying a feature.

5.2 Analysis on the representative behaviors

First, we show the characteristics of the groups generated from FLARE to check if our approach delivers useful insights. Figure 10 is the radar plot showing the characteristics of the 18 representative groups. We observe that the groups have distinct characteristics described in the high-level metric (i.e., PC) space. Note that some clusters look alike but actually have major differences in one or

Setup	Descriptions
Baseline	30MB LLC/socket, 1.2 - 2.9GHz clock, Hyperthreading enabled
Feature 1	**12MB LLC/socket**, 1.2 - 2.9GHz clock, Hyperthreading enabled
Feature 2	30MB LLC/socket, **1.2 - 1.8GHz clock**, Hyperthreading enabled
Feature 3	30MB LLC/socket, 1.2 - 2.9GHz clock, **Hyperthreading disabled**

Table 4: Summary of the datacenter-improving features.



(a) Comprehensive performance impact (MIPS reduction) on all HP jobs.

(b) Per-HP job performance impact (MIPS reduction).

Figure 12: Evaluating the performance estimation accuracy of FLARE.

more PCs. For example, Cluster 0 and Cluster 1 look similar but have major differences in PCs 0, 3, 6, 7 and 16.

Interestingly, there exist many clusters with similar weights around 10%. This indicates that our datacenter *does not* have a specific dominant characteristic. Instead, it is a collection of a wide variety of behaviors having similar importance. This emphasizes that a feature must be evaluated based on diverse datacenter behaviors for an accurate performance projection.

Figure 11 shows the performance impact of the three features measured from each group's representative scenarios. We note that the groups respond differently to a same feature due to their unique resource and performance characteristics.

Using the radar plots, we can also reason about why a specific scenario has certain performance impacts. For example, Cluster 8 has the biggest impact from Feature 1, which is to reduce the LLC capacity. Naturally, we can assume that Cluster 8 consists of scenarios with high LLC pressures. We find out that the key characteristics of Cluster 8 are high PC 12 value and low PC 7 value. According to our interpretation (Figure 8), PC 12 promotes the tendency to have more LLC misses, and negative PC 7 also promotes the LLC misses. Therefore, we can conclude that Cluster 8 is a group which would have greater sensitivity to LLC-related features.

As such, our representative scenarios successfully cover the diverse spectrum of datacenter behaviors and also allows us to perform deeper analysis without blindly reproducing a large number of scenarios.

5.3 Feature evaluation accuracy

All-job impact. First, we investigate how well FLARE estimates a feature's *overall* performance impact on the HP jobs. Figure 12a shows the performance impact measured from the whole datacenter (i.e., the true impact), a random sampling method, and FLARE. For the sampling, we randomly pick 18 job co-location scenarios (i.e., the same evaluation overheads as FLARE) and estimate the performance from them. We perform 1,000 sampling trials and show the resulting distribution with violin plots and box plots.

While sampling is generally an effective technique to embrace the variance, FLARE provides much more reliable estimations based on carefully selected representative scenarios. Our estimation errors are constantly low (<1% in absolute numbers) while sampling can cause errors up to 4%.

Another major benefit of FLARE is that we can always utilize the clusters' characteristics to reason about how we achieve such performance numbers (e.g., Section 5.2). In contrast, random sampling cannot provide any insights regarding its estimations.

Per-job impact. Next, we check how FLARE estimates the *per-job* performance impacts. Similar to the methodology introduced in Section 4.5, we may use the individual job performance in the representative scenarios to estimate a feature's impact on a specific job.

However, in this case, a representative scenario may *not include* the job of interest while its group actually has numerous instances of the job. In such a case, we check the next nearest (i.e., representative) scenario to the cluster center until we find the target job. The rest of the process is identical to the all-job performance estimation case, except that we weight the impacts by the number of job instances in that group (i.e., the likelihood to observe the job).

Figure 12b shows the performance estimation from the datacenter, sampling, and FLARE. This time, we use the 95% confidence interval for the sampling method. We do not provide the conventional load testing results as they have poor accuracy for Feature 1 (Figure 2).

Unlike the previous case, Sampling shows quite good performance for some selected workloads and features. We have two explanations for this behavior. First, certain jobs are robust to the resource contention and thus have inherently small variance. For the previous all-job performance case, since we add up the job performances for each scenario, the performance variances are much larger. Sampling from small-variance populations allows it to accurately estimate the impacts. Second, the per-job sampling population sizes are much smaller than that of the all-job case. For example, the sampling population size for the all-job impact estimation case is 895 (assuming every scenario has at least one HP job), whereas the population size for a per-job impact estimation case is usually less than half of that (i.e., many scenarios *do not* have the job of interest). Since the population size becomes smaller for the per-job case, we are more likely to get better quality samples.

In addition, we observe that FLARE occasionally makes inaccurate estimations. In fact, this is natural considering that we generate the clusters based on the general job and machine characteristics,

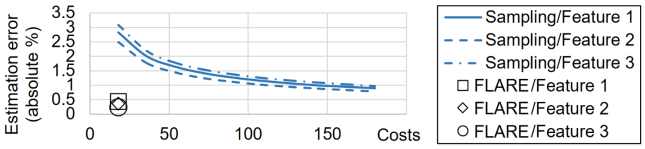


Figure 13: Comparing the evaluation costs of FLARE and random sampling.

not the per-job performance characteristics. We believe including the per-job metrics in our method would greatly improve the estimation accuracy for the job. However, excessively adding per-job metrics would increase the dimension of the feature space and may deteriorate the clustering quality. We therefore do not use any per-job metrics in deriving the representative scenarios, and recommend to include such metrics only when necessary.

5.4 Evaluation costs and accuracy

Lastly, we investigate the tradeoff between the evaluation costs and performance estimation fidelity. For FLARE, the evaluation cost is proportional to the number of representative scenarios (i.e., clusters). We observe that increasing the number of clusters *does not* improve the estimation quality, unless the number becomes very large. We therefore assume fixed cost and fidelity for FLARE.

In contrast, sampling can make better predictions as we spend more evaluation costs. Figure 13 shows the expected max performance estimation error for sampling (i.e., 95% confidence interval) and FLARE. Surprisingly, even with 10× the evaluation costs of FLARE, sampling cannot achieve similar quality results due to the large performance variance inherent in datacenter job co-location scenarios.

This shows that FLARE is a systematic and efficient approach to evaluate datacenter environments. We successfully evaluate the characteristics of 895 scenarios with only 18 representative behaviors, and achieve 50× reduction in the evaluation overheads.

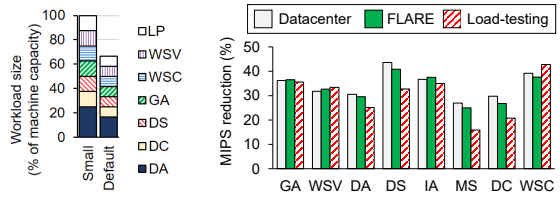
5.5 How to handle heterogenous configurations

Throughout the paper, we use FLARE to evaluate a datacenter with homogeneous machines, assuming that a feature would not change the machine's shape (e.g., core count and RAM capacity) and the corresponding job co-location scenarios. However, many real-world datacenters have *heterogeneous* machines and some features do change the machine's shape as well (e.g., new CPU). We therefore discuss how FLARE should be utilized in such environments. Table 5 shows the two machine types that we consider. We have added the *Small* type in addition to the default that we have used throughout the paper.

First, we try using the representative scenarios derived from the default machine for small machine evaluation. If the scenarios are

Resource	Default	Small
CPU	Intel Xeon E5-2650 v4 (2 sockets, 24 vCPUs/socket)	Intel Xeon E5-2640 v3 (2 sockets, 16 vCPUs/socket)
DRAM	256GB DDR4 2400MHz	128GB DDR4 2133MHz
Disk	Intel 730 SSD	Samsung 850 SSD
Network	Intel X710 10Gb	Intel 82599ES 10Gb

Table 5: Two Datacenter Configurations



(a) Job colocation change in heterogeneous machines

(b) Performance estimation results for the new (small) machine shape. We show per-job estimations for Feature 2.

Figure 14: Utilizing FLARE to handle a new machine shape.

truly representative and compatible, they should be able to tell the performance difference across different machine shapes just as they do for the other features.

However, we find that it is difficult to *exactly* reproduce a specific job co-location scenario on different shapes of machines. Figure 14a shows such an example. In this particular scenario, we have two 4 vCPU `DA` instances and one 4 vCPU `DC, DS, GA, WSC, WSV,` and `LP` instances. While this translates to ~70% utilization on the default machine, it becomes a full saturation scenario for the small machine. We may re-scale the job instance sizes (4 vCPU) to preserve the scenario's machine utilization, but this would alter the job' nature. In other words, it is impossible to reproduce the identical scenarios on different shapes of machines.

Nonetheless, thanks to its systematic method and generic applicability, we confirm that FLARE still provides a *new set* of representative scenarios if we apply it to the new machine shape. As shown in Figure 14b, we successfully estimate a feature's impact once the new representatives are derived from the new shape datacenter.

We therefore recommend to generate and utilize the representative scenarios *for each machine shape*. When comparing the machines with different shapes, there would be no single representative set which works across heterogeneous machines. However, if a user does not mind comparing the performance from different execution scenarios (e.g., embrace that the workload co-locations would differ by the machine shape and we have no control over it), he/she may scale and utilize the representative scenarios from one machine shape to evaluate other shapes, as discussed above.

We also emphasize that this per-machine type representative scenario approach is a sensible solution with reasonable overheads. In real datacenters, machine shape-changing features are infrequently introduced (e.g., once every year) but the existing machines last a few years until their retirements experiencing numerous feature upgrades (e.g., software updates). In other words, it is worth investing some efforts to extract representative scenarios for each machine shape as they will be utilized for 5-10 years.

5.6 How to handle scheduler changes

Changes in datacenter scheduler can make the representative scenarios from FLARE less valid as they tweak the job co-location scenarios. However, unlike the case where the machine's shape is changing (Section 5.5), the new schedulers do not generate whole new unseen scenarios. Instead, they promote more desirable scenarios (e.g., high utilization with less resource contention) and prohibit less desirable ones. Therefore, if we can get quick estimates of what

kind of job co-location scenarios would appear more (or less) for a new scheduler, we can use that information to derive new representative scenarios starting from Step 3 of Section 4 and 4. As the major overhead of FLARE comes from Step 1 (i.e., collecting enough job co-location scenarios), we can effectively handle new datacenter schedulers with minimal overheads.

6 RELATED WORK

Performance analysis and evaluation is a classic topic in the computer systems and architecture field. Accordingly, significant efforts have been put to reduce the analysis and evaluation overheads. We discuss the relationship between previous work and FLARE

Leveraging similarity in observations. One simple and effective optimization technique in performance evaluation is to skip the evaluation of redundant or similar data points. The idea has been extensively applied to reduce the simulation turnaround latency. Initial proposals [88, 98] successfully handled single-threaded applications, and later studies [1, 2, 9, 10, 34, 36, 56, 57] have extracted similarities from different scopes such as multithreaded applications, tasks and basic blocks, and pipeline behaviors.

FLARE also leverages the similarities in datacenter behaviors to reduce performance evaluation overheads. Our main contribution is to find the optimal representation of datacenter behaviors to maximize the reduction opportunities while preserving the original datacenter characteristics.

Acceleration by abstraction. Another group of studies gain evaluation speedup by abstracting away unnecessary details and focusing on the first-order performance constituents. Datacenter-scale simulators [54, 68, 91, 105] usually adopt abstract models (e.g., queues) to minimize the analysis loads. For FLARE, we abstract 100+ raw performance/resource metrics into a few high-level performance metrics to minimize analysis overheads. Also, our approach could be applied to the proposed abstract model to further optimize their evaluation overhead.

Characterizing applications. Application characterization studies have similar goals to the two categories discussed above. They first select the key metrics describing application behaviors and perform similarity analysis to eliminate redundancies. Many studies [41, 42, 73, 74, 79, 80] successfully characterize popular benchmark suites such as SPEC CPU [14] and MiBench [35] as well as find redundancies in input datasets [25, 26]. Some studies acknowledge the importance of co-locations [15, 18] but their major contribution is to accurately generate interference effects not to extract and reproduce production datacenter's behaviors.

FLARE has similar goals to these studies but *differs* in terms of the scope and the detailed methodology. Specifically, while previous studies focus on deduplicating particular benchmark suites, FLARE proposes a *generic* methodology to *systematically* extract the key metrics accurately describing job co-location scenarios of an arbitrary datacenter. To the best of our knowledge, we are the *first* to perform such an analysis on datacenters considering *job co-location scenarios* and *resource sharing behaviors*. Compared to the existing single application analyses, we also tackle a problem with greater amount of complexity.

Applying machine learning techniques. Machine learning techniques help reduce analysis and evaluation overheads by automatically identifying the key performance constituents. First, some studies [45, 46] predict performance using simple machine learning models trained on a large number of system behaviors instead of complex simulators. A state-of-the-art model further advances the technique to automatically detect datacenter performance issues [31]. Recent studies refine key performance-determining metrics [101, 103] from a large number raw observations using non-linear models. FLARE also adopts two well-validated [41, 80] and highly effective machine learning techniques – Principal Component Analysis (PCA) and K-means clustering – to systematically extract high-level performance metrics and group similar behaviors.

Datacenter analysis. Major industry players and researchers have analyzed datacenter behaviors to guide the development of new features [3, 5, 8, 23, 39, 48, 51, 55, 83]. Various techniques have been proposed to inspect large-scale distributed systems at application, library, and middleware levels [4, 29, 84, 89]. The analysis results successfully triggered the development of novel features [13, 48, 53, 63]. However, these analysis studies focus only on delivering lessons from large-scale environments, while FLARE provides an efficient performance evaluation method for datacenters.

Datacenter benchmarks. Datacenter benchmarking studies [6, 15, 18, 28, 30, 65, 72, 77, 96] successfully characterize and reproduce the nature of scale-out jobs running on datacenters and facilitate feature evaluation using more realistic scenarios. They thoroughly discuss the major differences from conventional jobs [7, 14] and have triggered successful development of novel system architectures which are highly optimized for datacenter jobs [61, 62].

While these studies provide valuable insights regarding individual datacenter jobs' traits, we discover that they *do not* thoroughly discuss how the jobs should be co-located to accurately estimate a feature's holistic impact on a datacenter. Some studies [15, 18, 30] acknowledge the importance of job co-location and model resource sharing behaviors in the benchmarks; however, their main contribution is to precisely generate the interferences or consider resource sharing effects, rather than defining how the representative co-location scenarios should be extracted from an arbitrary datacenter (i.e., FLARE's goal). In fact, we believe these benchmarks can help FLARE to reproduce specific resource sharing behaviors once we identify the representative job co-location scenarios. They are therefore orthogonal to our work.

7 CONCLUSION

This paper proposed FLARE, a Fast and Accurate Datacenter Evaluation methodology using representative system behaviors. We first characterized various datacenter system behaviors using systematically derived high-level metrics. We then extracted representative system behaviors, which effectively summarize the whole traits of the datacenter. We reproduced the selected behaviors with carefully designed load-testing benchmarks and accurately evaluated the performance impact of new features with minimal overheads. Our evaluation using an in-house datacenter and three different features showed that FLARE is 50× and 10×+ more efficient compared to full datacenter evaluation and sampling-based evaluation, while providing high accuracy (~1% errors).

ACKNOWLEDGMENTS

We thank Youngsok Kim in Yonsei University for helping us to improve the paper. This work was partly supported by SNU-SK Hynix Inc. Solution Research Center (S3RC), the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (NRF-2021R1F1A1062902, NRF-2020M3H6A1085527), and the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2021-0-02051) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).

REFERENCES

[1] Ehsan K Ardestani and Jose Renau. 2013. ESESC: A fast multicore simulator using time-based sampling. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 448–459.

[2] Eduardo Argollo, Ayose Falcón, Paolo Faraboschi, Matteo Monchiero, and Daniel Ortega. 2009. COTSon: infrastructure for full system simulation. *ACM SIGOPS Operating Systems Review* 43, 1 (2009), 52–61.

[3] Grant Ayers, Jung Ho Ahn, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Memory Hierarchy for Web Search. In *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*. IEEE, 643–656.

[4] Paul Barham, Rebecca Isaacs, and Dushyanth Narayanan. 2003. Magpie: online modelling and performance-aware systems. In *9th Workshop on Hot Topics in Operating Systems (HotOS-IX)*. USENIX.

[5] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. 2003. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro* 23, 2 (March 2003), 22–28.

[6] PerfKit Benchmarker. 2017. *PerfKit Benchmarker*. http://googlecloudplatform.github.io/PerfKitBenchmarker/

[7] Christian Bienia. 2011. *Benchmarking modern multiprocessors*.

[8] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Comput. Netw. ISDN Syst.* 30, 1-7 (April 1998), 107–117.

[9] T. E. Carlson, W. Heirman, and L. Eeckhout. 2013. Sampled simulation of multi-threaded applications. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2–12.

[10] T. E. Carlson, W. Heirman, K. Van Craeynest, and L. Eeckhout. 2014. BarrierPoint: Sampled simulation of multi-threaded applications. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2–12.

[11] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. 2017. Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. ACM, New York, NY, USA, 17–32.

[12] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. 2016. Baymax: QoS Awareness and Increased Utilization for Non-Preemptive Accelerators in Warehouse Scale Computers. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. ACM, New York, NY, USA, 681–696.

[13] Eric S. Chung, John D. Davis, and Jaewon Lee. 2013. LINQits: Big Data on Little Clients. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (Tel-Aviv, Israel) *(ISCA '13)*. ACM, New York, NY, USA, 261–272.

[14] Standard Performance Evaluation Corporation. 2006. *SPEC CPU2006*. https://www.spec.org/cpu2006/

[15] Standard Performance Evaluation Corporation. 2013. *SPEC virt_sc 2013*. https://www.spec.org/virt_sc2013

[16] 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) *(SOSP '17)*. ACM, New York, NY, USA, 153–167.

[17] Jeffrey Dean and Sanjay Ghemawat. 2010. MapReduce: A Flexible Data Processing Tool. *Commun. ACM* 53, 1 (Jan. 2010), 72–77.

[18] Christina Delimitrou and Christos Kozyrakis. 2013. iBench: Quantifying interference for datacenter applications. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*. 23–33.

[19] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, Texas, USA) *(ASPLOS '13)*. ACM, New York, NY, USA, 77–88.

[20] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. ACM, New York, NY, USA, 127–144.

[21] Christina Delimitrou and Christos Kozyrakis. 2016. HCloud: Resource-Efficient Provisioning in Shared Cloud Systems. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. ACM, New York, NY, USA, 473–488.

[22] Christina Delimitrou, Daniel Sanchez, and Christos Kozyrakis. 2015. Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (Kohala Coast, Hawaii) *(SoCC '15)*. ACM, New York, NY, USA, 97–110.

[23] Christina Delimitrou, Sriram Sankar, Kushagra Vaid, and Christos Kozyrakis. 2011. Decoupling datacenter studies from access to large-scale applications: A modeling approach for storage workloads. In *2011 IEEE International Symposium on Workload Characterization (IISWC)*. 51–60.

[24] L. Eeckhout, J. Sampson, and B. Calder. 2005. Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation. In *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005*. 2–12.

[25] Lieven Eeckhout, Hans Vandierendonck, and Koenraad De Bosschere. 2002. Workload design: Selecting representative program-input pairs. In *Parallel Architectures and Compilation Techniques, 2002. Proceedings. 2002 International Conference on*. IEEE, 83–94.

[26] Lieven Eeckhout, Hans Vandierendonck, and Koen De Bosschere. 2003. Quantifying the impact of input data sets on program behavior and its applications. *Journal of Instruction-Level Parallelism* 5, 1 (2003), 1–33.

[27] Stijn Eyerman and Lieven Eeckhout. 2008. System-level performance metrics for multiprogram workloads. *IEEE micro* 28, 3 (2008).

[28] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems* (London, England, UK) *(ASPLOS XVII)*. ACM, New York, NY, USA, 37–48.

[29] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. 2007. X-trace: A Pervasive Network Tracing Framework. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation* (Cambridge, MA) *(NSDI'07)*. USENIX Association, Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=1973430.1973450

[30] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 3–18.

[31] Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. 2019. Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 19–33.

[32] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. 2016. Altruistic Scheduling in Multi-resource Clusters. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, Berkeley, CA, USA, 65–80. http://dl.acm.org/citation.cfm?id=3026877.3026884

[33] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. 2016. GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 81–97. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/grandl_graphene

[34] Thomas Grass, Alejandro Rico, Marc Casas, Miquel Moreto, and Eduard Ayguadé. 2016. Taskpoint: Sampled simulation of task-based programs. In *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 296–306.

[35] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 3–14.

[36] Jaewon Lee Jangwoo Kim Hanhwi Jang, Jae-eon Jo. 2018. RpStacks-MT: A High-throughput Multi-core Processor Design Evaluation Methodology. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*

(MICRO-51).

[37] Johann Hauswald, Yiping Kang, Michael A. Laurenzano, Quan Chen, Cheng Li, Trevor Mudge, Ronald G. Dreslinski, Jason Mars, and Lingjia Tang. 2015. DjiNN and Tonic: DNN As a Service and Its Implications for Future Warehouse Scale Computers. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '15)*. ACM, New York, NY, USA, 27–40.

[38] Johann Hauswald, Michael A. Laurenzano, Yunqi Zhang, Cheng Li, Austin Rovinski, Arjun Khurana, Ron Dreslinski, Trevor Mudge, Vinicius Petrucci, Lingjia Tang, and Jason Mars. 2015. Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (ASPLOS '15)*. ACM, New York, NY, USA, 13 pages.

[39] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*. IEEE, 620–629.

[40] John L. Hennessy and David A. Patterson. 2011. *Computer Architecture, Fifth Edition: A Quantitative Approach* (5th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[41] Kenneth Hoste and Lieven Eeckhout. 2007. Microarchitecture-independent workload characterization. *IEEE micro* 27, 3 (2007).

[42] Kenneth Hoste, Aashish Phansalkar, Lieven Eeckhout, Andy Georges, Lizy K John, and Koen De Bosschere. 2006. Performance prediction based on inherent program similarity. In *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*. ACM, 114–122.

[43] Chang-Hong Hsu, Qingyuan Deng, Jason Mars, and Lingjia Tang. 2018. SmoothOperator: Reducing Power Fragmentation and Improving Power Utilization in Large-scale Datacenters. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 535–548.

[44] Chang-Hong Hsu, Yunqi Zhang, Michael A. Laurenzano, David Meisner, Thomas F. Wenisch, Jason Mars, Lingjia Tang, and Ronald G. Dreslinski. 2015. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 271–282.

[45] Engin Ïpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. 2006. Efficiently Exploring Architectural Design Spaces via Predictive Modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, California, USA) *(ASPLOS XII)*. Association for Computing Machinery, New York, NY, USA, 195–206.

[46] PJ Joseph, Kapil Vaswani, and Matthew J Thazhuthaveetil. 2006. A predictive performance model for superscalar processors. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 161–170.

[47] Ajay Joshi, Aashish Phansalkar, Lieven Eeckhout, and Lizy Kurian John. 2006. Measuring Benchmark Similarity Using Inherent Program Characteristics. *IEEE Trans. Comput.* 55, 6 (June 2006), 769–782.

[48] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 1–12.

[49] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, and Sriram Rao. 2016. Morpheus: Towards Automated SLOs for Enterprise Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 117–134. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/jyothi

[50] Melanie Kambadur, Tipp Moseley, Rick Hank, and Martha A Kim. 2012. Measuring interference between live datacenter applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 51.

[51] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a Warehouse-scale Computer. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '15)*. ACM, New York, NY, USA, 158–169.

[52] Svilen Kanev, Kim Hazelwood, Gu-Yeon Wei, and David Brooks. 2014. Tradeoffs between power management and tail latency in warehouse-scale applications. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. 31–40.

[53] Svilen Kanev, Sam Likun Xi, Gu-Yeon Wei, and David Brooks. 2017. Mallacc: Accelerating Memory Allocation. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. ACM, New York, NY, USA, 33–45.

[54] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, et al. 2018. Firesim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 29–42.

[55] Christos Kozyrakis, Aman Kansal, Sriram Sankar, and Kushagra Vaid. 2010. Server Engineering Insights for Large-Scale Online Services. *IEEE Micro* 30, 4 (July 2010), 8–19.

[56] Jaewon Lee, Hanhwi Jang, Jae-eon Jo, Gyu-hyeon Lee, and Jangwoo Kim. 2017. StressRight: Finding the right stress for accurate in-development system evaluation. In *Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on*. IEEE, 205–216.

[57] Jaewon Lee, Hanhwi Jang, and Jangwoo Kim. 2014. Rpstacks: Fast and accurate processor design space exploration using representative stall-event stacks. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 255–267.

[58] Jaewon Lee, Changkyu Kim, Kun Lin, Liqun Cheng, Rama Govindaraju, and Jangwoo Kim. 2018. WSMeter: A Performance Evaluation Methodology for Google's Production Warehouse-Scale Computers. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 549–563.

[59] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. 2009. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *Proceedings of the 36th Annual International Symposium on Computer Architecture* (Austin, TX, USA) *(ISCA '09)*. ACM, New York, NY, USA, 267–278.

[60] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '15)*. ACM, New York, NY, USA, 450–462.

[61] Pejman Lotfi-Kamran, Boris Grot, and Babak Falsafi. 2012. NOC-Out: Microarchitecting a scale-out processor. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 177–187.

[62] Pejman Lotfi-Kamran, Boris Grot, Michael Ferdman, Stavros Volos, Onur Kocberber, Javier Picorel, Almutaz Adileh, Djordje Jevdjic, Sachin Idgunji, Emre Ozer, and Babak Falsafi. 2012. Scale-out Processors. In *Proceedings of the 39th Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '12)*. IEEE Computer Society, Washington, DC, USA, 500–511. http://dl.acm.org/citation.cfm?id=2337159.2337217

[63] Martin Maas, Krste Asanović, and John Kubiatowicz. 2018. A hardware accelerator for tracing garbage collection. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 138–151.

[64] Jason Mars and Lingjia Tang. 2013. Whare-map: Heterogeneity in "Homogeneous" Warehouse-scale Computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (Tel-Aviv, Israel) *(ISCA '13)*. ACM, New York, NY, USA, 619–630.

[65] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture* (Porto Alegre, Brazil) *(MICRO-44)*. ACM, New York, NY, USA, 248–259.

[66] David Meisner, Brian T. Gold, and Thomas F. Wenisch. 2009. PowerNap: Eliminating Server Idle Power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems* (Washington, DC, USA) *(ASPLOS XIV)*. ACM, New York, NY, USA, 205–216.

[67] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. 2011. Power Management of Online Data-intensive Services. In *Proceedings of the 38th Annual International Symposium on Computer Architecture* (San Jose, California, USA) *(ISCA '11)*. ACM, New York, NY, USA, 319–330.

[68] David Meisner, Junjie Wu, and Thomas F Wenisch. 2012. Bighouse: A simulation infrastructure for data center systems. In *2012 IEEE International Symposium on Performance Analysis of Systems & Software*. IEEE, 35–45.

[69] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 330–339.

[70] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.

[71] Khang T Nguyen. 2017. *Introduction to Cache Allocation Technology in the Intel Xeon Processor E5 v4 Family*. https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[72] Tapti Palit, Yongming Shen, and Michael Ferdman. 2016. Demystifying cloud benchmarking. In *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 122–132.

[73] Reena Panda and Lizy Kurian John. 2017. Proxy benchmarks for emerging big-data workloads. In *Parallel Architectures and Compilation Techniques (PACT), 2017 26th International Conference on*. IEEE, 105–116.

[74] Reena Panda, Shuang Song, Joseph Dean, and Lizy K John. 2018. Wait of a Decade: Did SPEC CPU 2017 Broaden the Performance Horizon?. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 271–282.

[75] David A. Patterson. 2008. Technical Perspective: The Data Center is the Computer. *Commun. ACM* 51, 1 (Jan. 2008), 105–105.

[76] Steven Pelley, David Meisner, Pooya Zandevakili, Thomas F. Wenisch, and Jack Underwood. 2010. Power Routing: Dynamic Power Provisioning in the Data Center. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems* (Pittsburgh, Pennsylvania, USA) *(ASPLOS XV)*. ACM, New York, NY, USA, 231–242.

[77] Transaction Processing performance Council. 2024. *TPC-Homepage*. http://www.tpc.org

[78] Vinicius Petrucci, Michael A. Laurenzano, John Doherty, Yunqi Zhang, Daniel Mosse, Jason Mars, and Lingjia Tang. 2015. Octopus-Man: QoS-driven task management for heterogeneous multicores in warehouse-scale computers. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 246–258.

[79] Aashish Phansalkar, Ajay Joshi, Lieven Eeckhout, and Lizy Kurian John. 2005. Measuring program similarity: Experiments with SPEC CPU benchmark suites. In *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*. IEEE, 10–20.

[80] Aashish Phansalkar, Ajay Joshi, and Lizy K. John. 2007. Analysis of Redundancy and Application Balance in the SPEC CPU2006 Benchmark Suite. In *Proceedings of the 34th Annual International Symposium on Computer Architecture* (San Diego, California, USA) *(ISCA '07)*. ACM, New York, NY, USA, 412–423.

[81] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 7.

[82] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. 2011. *Google cluster-usage traces: format + schema*. Technical Report. Google Inc., Mountain View, CA, USA. Revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

[83] Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, and Robert Hundt. 2010. Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers. *IEEE Micro* 30, 4 (July 2010), 65–79.

[84] Patrick Reynolds, Charles Killian, Janet L. Wiener, Jeffrey C. Mogul, Mehul A. Shah, and Amin Vahdat. 2006. Pip: Detecting the Unexpected in Distributed Systems. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3* (San Jose, CA) *(NSDI'06)*. USENIX Association, Berkeley, CA, USA, 9–9. http://dl.acm.org/citation.cfm?id=1267680.1267689

[85] Lior Rokach and Oded Maimon. 2005. Clustering methods. In *Data mining and knowledge discovery handbook*. Springer, 321–352.

[86] Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.

[87] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. 2013. Omega: Flexible, Scalable Schedulers for Large Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (Prague, Czech Republic) *(EuroSys '13)*. ACM, New York, NY, USA, 351–364.

[88] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. 2002. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, California) *(ASPLOS X)*. ACM, New York, NY, USA, 45–57.

[89] Benjamin H Sigelman, Luiz Andre 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. Technical report, Google.

[90] Akshitha Sriraman and Thomas F Wenisch. 2018. μ Suite: A Benchmark Suite for Microservices. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–12.

[91] Zhangxi Tan, Zhenghao Qian, Xi Chen, Krste Asanovic, and David Patterson. 2015. DIABLO: A warehouse-scale computer network simulator using FPGAs. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 207–221.

[92] Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. 2011. The Impact of Memory Subsystem Resource Sharing on Datacenter Applications. In *Proceedings of the 38th Annual International Symposium on Computer Architecture* (San Jose, California, USA) *(ISCA '11)*. ACM, New York, NY, USA, 283–294.

[93] Lingjia Tang, Jason Mars, Xiao Zhang, Robert Hagmann, Robert Hundt, and Eric Tune. 2013. Optimizing Google's warehouse scale computers: The NUMA experience. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. 188–197.

[94] D. M. Tullsen, S. J. Eggers, and H. M. Levy. 1995. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings 22nd Annual International Symposium on Computer Architecture*. 392–403.

[95] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems* (Bordeaux, France) *(EuroSys '15)*. ACM, New York, NY, USA, Article 18, 17 pages.

[96] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, Chen Zheng, Gang Lu, Kent Zhan, Xiaona Li, and Bizhu Qiu. 2014. BigDataBench: A big data benchmark suite from internet services. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 488–499.

[97] John Wilkes. 2011. More Google cluster data. Google research blog. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html..

[98] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. 2003. SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling. In *Proceedings of the 30th Annual International Symposium on Computer Architecture* (San Diego, California) *(ISCA '03)*. ACM, New York, NY, USA, 84–97.

[99] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. 2013. Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (Tel-Aviv, Israel) *(ISCA '13)*. ACM, New York, NY, USA, 607–618.

[100] Ahmad Yasin. 2014. A top-down method for performance analysis and counters architecture. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 35–44.

[101] Qinyi Luo Zhibin Yu Xuehai Qian Yirong Lv, Bin Sun. 2018. CounterMiner: Mining Big Performance Data from Hardware Counters. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-51)*.

[102] Z. Yu, J. Wang, L. Eeckhout, and C. Xu. 2018. QIG: Quantifying the Importance and Interaction of GPGPU Architecture Parameters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 6 (June 2018), 1211–1224.

[103] Zhibin Yu, Wen Xiong, Lieven Eeckhout, Zhendong Bei, Avi Mendelson, and Chengzhong Xu. 2018. Mia: Metric importance analysis for big data workload characterization. *IEEE Transactions on Parallel and Distributed Systems* 29, 6 (2018), 1371–1384.

[104] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. 2013. CPI2: CPU Performance Isolation for Shared Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (Prague, Czech Republic) *(EuroSys '13)*. ACM, New York, NY, USA, 379–391.

[105] Yanqi Zhang, Yu Gan, and Christina Delimitrou. 2019. μqSim: Enabling Accurate and Scalable Simulation for Interactive Microservices. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 212–222.

[106] Yunqi Zhang, David Meisner, Jason Mars, and Lingjia Tang. 2016. Treadmill: Attributing the Source of Tail Latency Through Precise Load Testing and Statistical Inference. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 456–468.

[107] Yunqi Zhang, George Prekas, Giovanni Matteo Fumarola, Marcus Fontoura, Inigo Goiri, and Ricardo Bianchini. 2016. History-Based Harvesting of Spare Cycles and Storage in Large-Scale Datacenters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 755–770. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/zhang-yunqi

[108] Haishan Zhu and Mattan Erez. 2016. Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. ACM, New York, NY, USA, 33–47.