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

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

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.

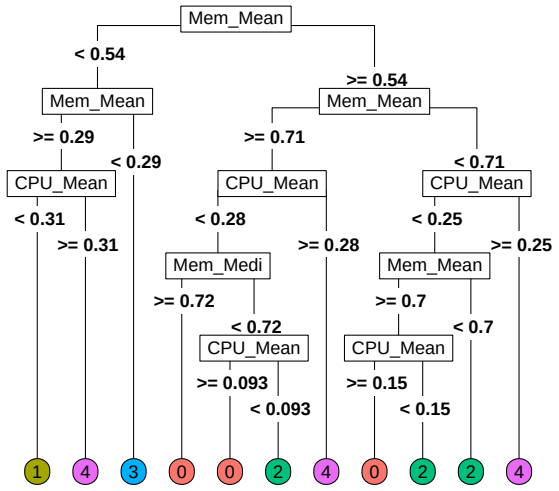

Figure 1: Explanation of the clustering.

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

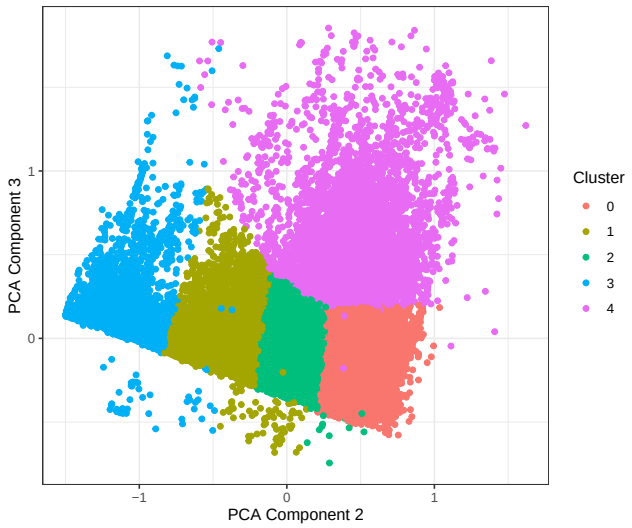

Figure 2: Visualization of the clustering.

clusters over the selected periods in time. To this end, we only consider containers that exist across all periods in time.

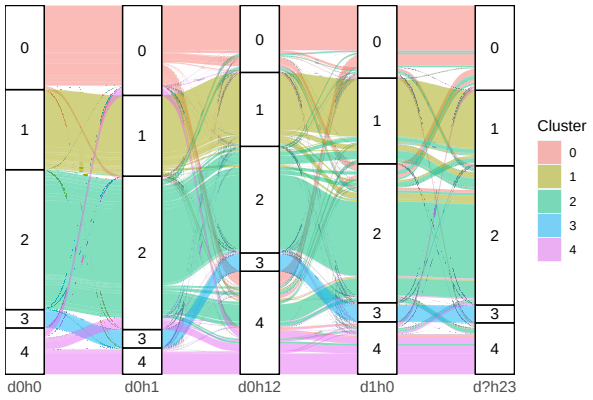

Figure 3: Visualization of the temporal performance deviation through the shift of microservices between clusters over time. The color of each flow represents the initial cluster.

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