# Analyzing Performance Variability in Alibaba's Microservice Architecture: A Critical-Path-Based Perspective

Alireza Ezaz
Brock University
St. Catharines, Ontario, Canada
sezaz@brocku.ca

Ghazal Khodabandeh
Brock University
St. Catharines, Ontario, Canada
gkhodobandeh@brocku.ca

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

## ABSTRACT

In large-scale microservice architectures, such as those utilized by Alibaba, identifying and addressing performance bottlenecks is a significant challenge due to the complicated interactions between thousands of services. To navigate this challenge, we have developed a critical-path-based technique aimed at analyzing microservice interactions within these complex systems. This technique facilitates the identification of critical nodes where service requests experience the longest delays. Our contribution is the discovery of performance variability in service interactions' response times within these critical paths, and pinpointing specific interactions within the system that show a high degree of performance variability. This improves the ability to detect service performance issues and their root causes allowing for dynamic adjustment in data collection detail, and targets critical interactions for adaptive monitoring.

## CCS CONCEPTS

• **General and reference** → **Performance**; • **Software and its engineering** → *Software maintenance tools*; • **Applied computing** → **Service-oriented architectures**.

## KEYWORDS

Microservice Architecture, Performance Bottlenecks, Critical Path, Response Time Variability, Adaptive Tracing

**ACM Reference Format:**
Alireza Ezaz, Ghazal Khodabandeh, and Naser Ezzati-Jivan. 2024. Analyzing Performance Variability in Alibaba's Microservice Architecture: A Critical-Path-Based Perspective. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3629527.3651845

## 1 INTRODUCTION

Distributed tracing is crucial for tracking how applications perform across a system, ensuring that operations are smooth and reliable. Unlike tools that focus on individual components, distributed tracing monitors entire requests as they traverse various parts of the system architecture, from a user's click to data storage in a database [9]. This monitoring plays an important role in identifying where issues begin, spotting any behavior that does not align with the expected operation of the system, detecting deviations in performance and finally enhancing the system's overall effectiveness.

However, the challenge intensifies when managing large-scale microservice-based applications like those at Alibaba, Uber, and Amazon, where thousands of services are constantly interacting. The complexity of these systems increases by complicated dependencies among services, the large number of monitoring metrics (e.g., Netflix exposes 2 million metrics and Uber exposes 500 million metrics [10]), and frequent updates, all alongside the extensive data produced by distributed tracing [11]. These factors make it particularly difficult to diagnose performance issues and pinpoint their root causes. To effectively manage these challenges, accurate detection of performance issues and clever analysis are essential.

Recent studies such as [6, 7, 12] have highlighted advancements in microservices monitoring and analysis, employing scalable, real-time frameworks and delving into microservice dependencies and performance. These contributions emphasize the utility of machine learning in predicting usage patterns and the importance of understanding microservice dependencies for improved performance analysis. However, a critical gap remains in the analysis of performance variability across microservices as those approaches often overlook the insights that can be gained from this analysis across different system components.

This is where we take performance variation analysis into account. By looking at how performance values fluctuate, we can find patterns or issues that point to the root of the problems. Consider a scenario in which a single microservice is involved in handling (a part of) three requests within a span of three minutes, but its latency varies significantly for each request. The first request is processed in just 1 millisecond, the second takes longer, at 30 milliseconds, and the third request experiences a substantial delay, requiring 500 milliseconds to complete. Such a notable difference in the service time of the same microservice, known as performance variability, if persistent across numerous interactions, is a red flag that indicates a potential problem warranting further investigation.

Building on this idea, our contribution is conducting a deeper analysis of the concept of performance variability in a large-scale microservice architecture. In our proposed technique, we begin with extracting critical paths from the trace dataset. A critical path is defined as a sequence of service interactions where requests experience the longest delays. After extracting critical paths from the requests, we group them into categories based on how similar they are to each other. For each group of requests sharing the same critical path, our technique focuses on extracting microservice


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

ACM ISBN 979-8-4007-0445-1/24/05…$15.00
https://doi.org/10.1145/3629527.3651845

interactions within these paths. It involves collecting response times for each interaction from the caller (upper) microservice to the callee (downstream) service over fixed time intervals. Then we calculate average response times and their variance for these interactions. By examining the collected data, our technique aims to identify abnormally high or low response time variations. This step is crucial for pinpointing specific microservices that deviate from expected performance norms, suggesting potential areas of concern. Such analysis uncovers deeper issues, from resource constraints to inefficient microservice executions, offering a refined approach to diagnosing and optimizing microservice issues.

## 2 RELATED WORK

Past efforts introduce various frameworks and techniques for monitoring microservice architecture and analysis of dependencies by creating call graphs and evaluating them for anomaly detection, root cause analysis, and further system optimization. However, they often overlook the insights that can be gained from performance variability analysis. Luo et al. [6] present a comprehensive study of large-scale microservice deployments in AliBaba's production clusters, focusing on the structural properties of microservice call graphs and call dependencies. It also offers an in-depth characterization of microservice runtime performance, providing insights into scheduling and resource management. Barham et al. [3] introduce Magpie, a tool designed to model and analyze system workloads by capturing resource consumption and control paths of requests in a system. It features an approach for detailed workload characterization without requiring modifications to the system, enabling accurate performance analysis and debugging. Similarly, Thalheim et al. [10] designed Sieve to derive actionable insights from monitored metrics in distributed systems. Sieve features a metrics reduction framework and a metrics dependency extractor, which together help in filtering out unimportant metrics and inferring metrics dependencies, thereby enhancing the management of microservices-based applications analysis. Other studies combine machine learning techniques with their designed frameworks to find interesting results in microservice systems. chen et al. [4] introduce a deep learning approach to automate fault diagnosis with high precision, to detect faults, and identify root causes effectively. In another study, Janecek et al. [5] introduced a framework for detecting performance anomalies in software systems through the analysis of system-level trace data by employing critical path analysis and machine learning clustering. Nevertheless, these studies focus on anomaly detection and system monitoring, bypassing an in-depth exploration of performance variability and its implications for system optimization.

## 3 METHODOLOGY

Building upon the foundational work of Ates et al. [2] and Sambasivan et al. [8], our methodology leverages the premise that performance variation is indicative of unknown system behaviors and that requests following similar workflows are expected to yield similar runtime footprints. Our study utilizes data from Alibaba's production clusters[1] [1], focusing specifically on the first one hour of the dataset. This initial period encompasses traces across nearly twenty thousand microservices, offering a concise yet significant observation window. The data includes service IDs within the call graph, performance metrics such as response time, and the relationships between upstream (caller) and downstream (callee) microservices, preparing the basis for our analysis.

Our proposed methodology, as shown in Fig 1, involves data preprocessing, critical path extraction, performance variation analysis within the critical paths, and visualization tools. Our developed tool and scripts used in this methodology are available online at [2]. In the subsequent sections, we will elaborate on each step separately.

### 3.1 Data Collection and Preprocessing

As shown in stage (1) of Fig 1, our method begins with the extraction and preprocessing of the trace data. The preprocessing phase is important for ensuring data integrity and usability. The following steps outline our preprocessing approach:

(1) **Dataset Cleaning:** We cleaned the dataset by removing entries with invalid or non-numeric values for response times.
(2) **Invalid Trace ID Removal:** We identified and excluded records associated with invalid response time values by isolating unique trace IDs corresponding to these records.
(3) **Null Value Handling:** Further cleaning involved discarding records containing any empty fields.
(4) **Unnecessary Features Removal:** We also removed unnecessary features to focus the dataset on essential metrics relevant to our study such as timestamp, trace ID, um (upper microservice), dm (downstream microservice), rt (response time of dm to um).

### 3.2 Critical Path Extraction

Following the preprocessing of Alibaba's microservice interaction data, our methodology advances to stage (2) in Fig 1; extraction of critical paths. The extraction process is detailed as follows:

(1) **Critical Path Identification:**
At this stage, we identify each request in AliBaba's trace dataset by filtering unique trace IDs. For each request, records are sorted by timestamp to establish the sequence of interactions. The 'endtime' for each interaction is calculated by adding its response time to its timestamp. The interaction with the longest 'endtime' signifies the end of the critical path. Tracing back from this endpoint, the sequence of services involved in the critical path, from upstream to downstream, is determined.
(2) **Grouping Requests Based on Critical Path Similarity:** Under the assumption that requests with similar workflows should yield similar performance metrics [2], our designed technique groups requests sharing an exact similar critical path.

By focusing on these critical paths extracted , we aim to identify the key areas where performance optimization efforts should be concentrated.

[1] https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2022

[2] https://github.com/Alireza-Ezaz/Analyzing-Performance-Variability-in-Alibaba-s-Microservice-Architecture

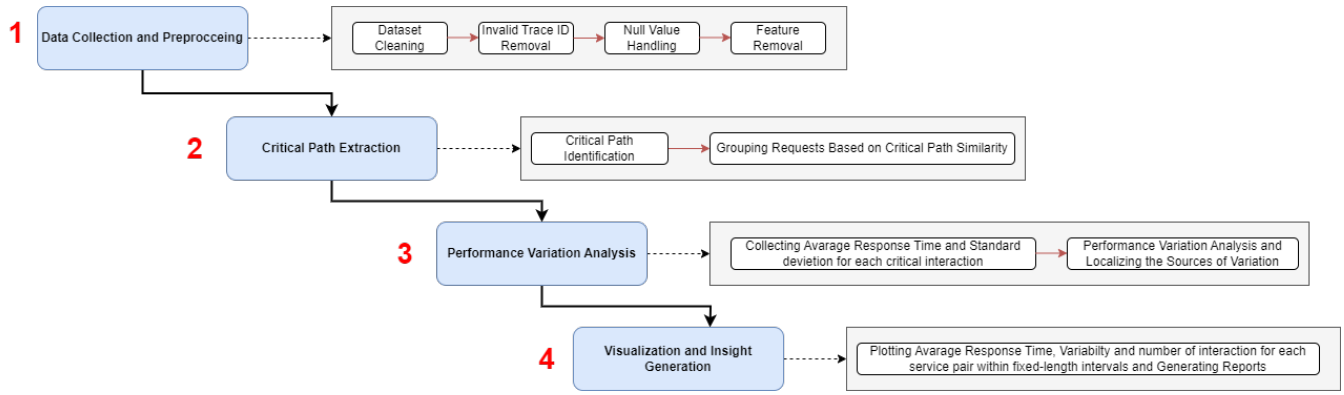


**Figure 1: The Architecture of Our Proposed Technique**

## 3.3 Performance Variation Analysis

Moving on to stage (3) in Fig 1, we investigate the performance variation within each group of requests with the same critical path. We then focus on how each specific microservice involved in the path, exhibit variability in its response times.

*3.3.1 Collecting Average Response Time and Standard deviation for each critical interaction.* For groups of requests following the same critical path, our method records the response times of each microservice interaction within these paths. This is done over twenty 3-minutes intervals creating a 1-hour observation window. After collecting these response times, it calculates the average response times and standard deviations, for each of the interactions, aiming to further localize the performance issue into a specific critical interaction in the next step.

*3.3.2 Performance Variation Analysis and Localizing the Sources of Variation.* As the core of our contribution, We analyze the variability in response times for each interaction within the extracted critical paths. Having the average and standard deviation for each interaction from the previous step, the average provides a baseline of expected performance, while the standard deviation reveals the extent of variability, offering insights into the consistency of the corresponding microservice's response. This dual metric approach enables a more granular understanding of performance variations, uncovering more details about microservice interactions. High variability (a large standard deviation compared to the average response time) suggests potential areas of concern, signaling that the performance of certain interactions (critical interactions) within the critical path is inconsistent and the problem can be localized to those specific critical interactions. This inconsistency may be indicative of deeper issues such as network latency, resource contention, or issues in service dependencies, which could adversely affect the overall system performance.

## 3.4 Visualization and Insight Generation

For further analysis in stage (4) of Fig 1, we plot the average response times and standard deviations across predetermined three-minute intervals, highlighting interactions that exhibit significant variability. These plots are helpful in pinpointing areas of instability, suggesting where further optimization and investigation are necessary. For each interaction characterized by high variability (our criteria for 'high'—selects interactions where the standard deviation exceeds ten times the mean response time), we generate dual-axis plots that represent the average response times against the occurrence counts over the intervals. At the end, we generate a report summarizing insights, including the number of unique traces examined, the variety of critical paths identified, and interactions with notable performance variability. This summary, alongside detailed statistics and visualizations, is compiled into user-friendly formats, ensuring that the insights are accessible to stakeholders involved in system development and optimization.

# 4 ANALYSIS AND DISCUSSION

Our investigation into the performance variation within Alibaba's microservice architecture led to the extraction of 91,704 unique critical paths from a 1-hour interval dataset of 40,062,862 requests, each distinguished by a unique trace ID. Of these, 1,891 microservice interactions within critical paths exhibited a noticeably high variance in performance. Our analysis has unfolded insightful patterns of response time variability. Delving into the data, we have presented twelve selected plots in Fig 2, each uncovering a critical interaction within the corresponding critical path. Our analysis is based on interactions grouped into four general patterns, characterized by abnormal fluctuations within at least one three-minute interval in an hour.

The analysis of data through our technique is visually represented in Fig 2, where the x-axis segments the observation window into 3-minute intervals. Green bars graphically depict the average response time per interval, with numerical standard deviation values indicated above, offering a clear view of performance fluctuations. The blue line complements this by illustrating interaction counts, and black error bars extend from each green bar to represent the range of response time variability, providing an overview of the interaction's performance variability. The presence of error bars extending into negative values does not imply negative response times; they simply show that the lower range of variability falls below the mean due to the subtraction of the standard deviation from the average. Interaction counts should be interpreted with reference to the right y-axis, and average response times should be related to the left y-axis.

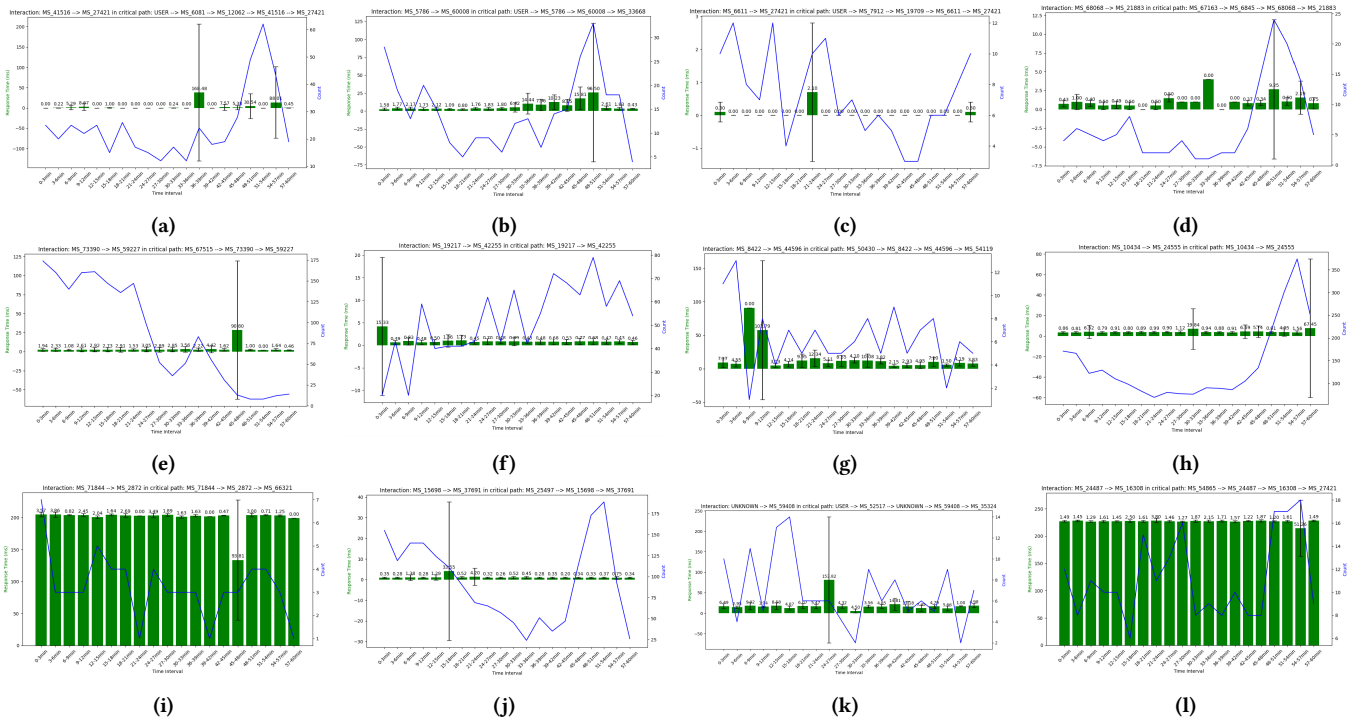

**Figure 2: Performance Variability Analysis for Different Critical Interactions within Corresponding Critical Paths**

Among the patterns observed, one notable scenario in Figures 2a, 2b, 2c, and 2d showcased some periods where the interaction count, average response time, and the variation in response time were simultaneously high. This indicates a potential overload scenario, suggesting the system was handling a higher volume of requests

Conversely, we also identified intervals like the first 30 minutes of Fig 2e where, despite a high number of interactions, the average response time and standard deviations remained stable, hinting at the system's ability to efficiently manage load without compromising on performance consistency.

Another pattern such as 36-39 minute interval in Fig 2a, 45-48 minute interval in Fig 2e, 0-3 minute interval in Fig 2f, 30-33 minute interval in Fig 2g, and 0-3 minute interval in Fig 2h, highlighted through our analysis was the occurrence of intervals where an increase in the average response time was accompanied by substantial variability, despite a lower interaction count compared to peak periods. This indicates that factors other than the volume of interactions, such as resource allocation or network issues, could be impacting performance.

Furthermore, we observed a critical interaction in 45-48 minute interval in Fig 2i where the average response time either decreased or remained low, yet the variability in response times increased. This suggests a growing inconsistency in how requests are processed, with some being completed swiftly while others face delays, leading to an unpredictable performance landscape.

In Figures 2j, 2k, and 2l we see a combination of the above scenarios of system performance previously discussed, each highlighting different ways that performance can vary under various conditions. Its analysis emphasizes the complex nature of performance issues, showing how different factors can impact the system's effectiveness and dependability. By analyzing these figures, we get a deeper insight into the system's behavior, which helps identify specific areas (critical interactions) that could benefit from optimization or further detailed study.

Overall, our results indicate that our approach provides a targeted method for monitoring and diagnosing the system's behavior and directs developers toward potential points of optimization, thereby mitigating system failures or inefficiencies. It also identifies critical interactions with high performance variation as prime candidates that can be considered for adaptive tracing and monitoring.

## 5 CONCLUSIONS AND FUTURE WORK

We analyzed performance variations within Alibaba's microservice architecture, focusing on identifying critical paths and analyzing response time variability. Our findings emphasize the complexity of managing performance in microservice architectures and the importance of continuous monitoring and analysis to identify and address bottlenecks.

Future work will extend this analysis by incorporating additional performance metrics such as CPU and memory utilization, applying machine learning algorithms to predict potential performance bottlenecks, and enhancing trace grouping techniques to efficiently manage and analyze large datasets. These efforts aim to improve the understanding and management of system performance, leading to more robust and efficient microservice architectures.

# REFERENCES


[1] Alibaba Group. 2022. Alibaba Cluster Data - Cluster Trace of Microservices. https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2022. Accessed: YYYY-MM-DD.

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

[3] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. 2004. Using Magpie for request extraction and workload modelling.. In *OSDI*, Vol. 4. 18–18.

[4] Hao Chen, Kegang Wei, An Li, Tao Wang, and Wenbo Zhang. 2021. Trace-based intelligent fault diagnosis for microservices with deep learning. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 884–893.

[5] Madeline Janecek, Naser Ezzati-Jivan, and Seyed Vahid Azhari. 2021. Container workload characterization through host system tracing. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 9–19.

[6] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*. 412–426.

[7] Barakat Saman. 2017. Monitoring and analysis of microservices performance. *Journal of Computer Science and Control Systems* 10, 1 (2017), 19.

[8] Raja R Sambasivan and Gregory R Ganger. 2012. Automated diagnosis without predictability is a recipe for failure. In *4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 12)*.

[9] Yuri Shkuro. 2019. *Mastering Distributed Tracing: Analyzing performance in microservices and complex systems*. Packt Publishing Ltd.

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

[11] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. 2020. Microrca: Root cause localization of performance issues in microservices. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–9.

[12] Cathy H Zhang and M Omair Shafiq. 2022. A Real-time, Scalable Monitoring and User Analytics Solution for Microservices-based Software Applications. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 6125–6134.