ClearCausal: Cross Layer Causal Analysis for Automatic Microservice Performance Debugging

Olufogorehan Tunde-Onadele* § fogo@insightfinder.com
Feiran Qin† fqin2@ncsu.edu
Xiaohui Gu† xgu@ncsu.edu
Yuhang Lin‡ § yuhangl@meta.com
*InsightFinder, Inc.
†North Carolina State University.
‡Meta, Inc.

***Abstract*—It is notoriously hard to debug performance problems in dynamic distributed systems such as containerized microservices. Developers need to spend large amounts of manual efforts to troubleshoot those performance bugs, which often incur high cost in terms of time and resources. In this paper, we present ClearCausal, a cross-layer causal analysis framework for microservices which performs *fine-grained* causal analysis between application function traces and infrastructure metric data to pinpoint both the root cause service and the root cause function. ClearCausal combines information theory-based causal inference anomaly detection, and dynamic dependency graph to achieve highly accurate root cause analysis. We have implemented ClearCausal and evaluated it using real performance bugs triggered in three benchmark microservices. Our results show that ClearCausal can accurately localize the buggy services and functions with much higher accuracy than other alternative solutions. Specifically, in all the tested bugs, ClearCausal successfully ranks the root cause service as the top candidate and ranks the root cause function in the top two. ClearCausal is lightweight, which makes it practical for real world production environments.**

***Index Terms*—Performance Debugging, Microservices**

I. Introduction

Microservice architectures have become a popular method for deploying applications in production cloud environments. Microservices divide applications into modular services that are responsible for different functionality and can efficiently scale to user demands. However, as microservice applications develop and become increasingly complex, debugging distributed performance bugs becomes a key challenge [1], [2]. The increased complexity makes it more difficult to understand the dependencies among different system components and pinpoint the root cause of a performance bug manifested in the production environment. For example, Microsoft suffered a recent outage in its microservices-supported 365 online web applications due to the performance issues in its caching infrastructure [3]. An automatic root cause analysis tool can guide developers to the root cause function quickly, saving debugging time and cost.

Much work (e.g., [4]–[6]) has been done to debug performance problems in distributed systems. However, highly dynamic microservice applications bring new challenges to this notoriously difficult problem. First, microservices often share a common physical host infrastructure for saving resources. As a result, many dependent or independent application pods/containers are co-located together, which makes the root cause analysis challenging. Previous work has proposed various correlation methods (e.g., Pearson [7]) to localize problematic microservice components [8]–[11]. However, those correlation approaches often suffer from high false positives, which sometimes incur more debugging work to the developer. Furthermore, previous work often cannot localize to the buggy function level, which still leaves most debugging work to the developer. Previous work also proposed to leverage Granger causality methods [6] to pinpoint root cause functions. However, they often do not consider the dynamic dependency relationships among different service components. Recent work also proposed machine learning-based methods to debug performance problems in microservices [12]–[14]. However, those approaches either require *labeled* training data or a large amount of historical data produced by a similar problem to train their models, which can only apply to previously seen performance problems.

A. Contribution

In this paper, we present ClearCausal, a new *dependency-aware cross-layer causal analysis framework*, which leverages both infrastructure-layer metrics (e.g., CPU usage, memory usage, network traffic) and application-layer performance traces to achieve *fine-grained* function-level performance debugging in distributed microservice applications. When a performance problem such as service level objective (SLO) violation and software hang is detected with existing methods [15], [16], ClearCausal aims at automatically localizing the root cause service and the buggy function among a large number of services and functions to help developers troubleshoot microservice performance issues quickly.

The key element that decides the effectiveness of ClearCausal is a robust causal inference method. Previous approaches have been focusing on applying *linear correlation* methods such as Pearson [7] over *raw* monitoring data (e.g., infrastructure metrics, application performance traces) directly. However, microservices are highly dynamic distributed systems. The correlations between different monitoring data are often non-linear. Moreover, raw monitoring data are often quite noisy and have time lags between each other.

To address those challenges, we propose a new hybrid approach that combines non-linear correlations and anomaly detection into a more robust causal inference scheme. We first employ *mutual information* (MI) [17], which observes Granger causality to discover linear or non-linear causal probabilities among different infrastructure or application perfor-

§Authors contributed while PhD students at North Carolina State University.

mance metrics. Second, instead of performing causal inference over raw monitoring data, we perform anomaly detection over both infrastructure metrics and application performance traces to extract principal patterns, which can effectively reduce the noise in the raw monitoring data for achieving robust causal inference. Since microservice systems often consist of many inter-dependent services, we also incorporate the service dependency in our causal analysis to surface the root cause service more effectively.

After localizing the faulty service, ClearCausal analyzes the function execution trace of the faulty service to localize the buggy function using a similar causal inference process. Because ClearCausal is triggered by performance anomaly alerts, it only performs on-demand analysis over a small window (e.g., 10 minutes) of recent trace data for root cause analysis.

ClearCausal enables autonomous performance debugging. Upon receiving performance alerts, ClearCausal performs anomaly detection and causal analysis without human intervention. ClearCausal can adapt itself to external feedback, which, however, is out of the scope of this paper.

Specifically, this paper makes the following contributions:

- We introduce a new dependency-aware cross-layer causal analysis system to achieve fine-grained function-level root cause localization for microservice systems.
- We present a robust hybrid causal inference algorithm to surface the root cause service and function with high accuracy without requiring a large amount of historical data for model training. Thus, our approach can be applied to previously known or unknown performance problems.
- We have implemented a prototype of ClearCausal[1] and evaluated it using a set of real world performance bugs injected in a set of microservice benchmark applications. Our results show that ClearCausal can accurately pinpoint the root cause service and function in all buggy runs with much higher accuracy than other alternative solutions. ClearCausal effectively ranks the root cause service as the top candidate and the root cause function in the top two for all the tested bugs. ClearCausal is light-weight, which imposes less than 3% overhead to the application.

The rest of the paper is structured as follows. Section II provides the system background. Section III presents the system design. Section IV describes our experimental evaluation. Section V compares our work with related work. Section VI concludes this paper.

II. Background

In this section, we discuss the preliminary components that support ClearCausal.

A. Data Collection

To achieve fine-grained root cause analysis, we perform cross-layer data collection for each microservice from both infrastructure layer (e.g., CPU usage, memory usage, disk usage, network bandwidth) and application performance layer (e.g., service and function execution time). In order to extract fine-grained function and performance metrics (e.g., response time time series, function execution time series), we leverage the trace data provided by open-source tools [18], [19]. Querying the tools with a certain start time and end time returns all the traces that are produced during the time window. Each trace consists of multiple spans that offer timing information of each operation performed as a request executes within a service.

[1]ClearCausal is released as open source software at https://github.com/NCSU-DANCE-Research-Group/ClearCausal.

B. Anomaly Detection

Microservice environments deal with the challenge of rapidly changing workloads. As such, the system metrics and spans are also dynamic time series data. However, the fluctuations over extended periods is a problem for our causal analysis computations. We observe that the noise can lead to misleading MI calculation results.

Our anomaly detection leverages an unsupervised machine learning method called self-organizing map (SOM) [20]. We chose SOM for its robustness of handling many noisy metrics from a large number of microservices, which also supports real-time online anomaly detection. SOM projects each multi-dimensional input data onto a reduced two-dimensional map of neurons. We identify anomalies based on the algorithm proposed by Dean et al. [16]. We empirically determine the neighborhood size threshold to detect anomalies as 85 percentile, the same percentile used in the original work. We also filter out transient anomalies close to the training data mean.

III. System Design

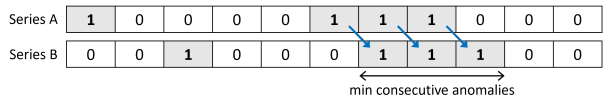
In this section, we present the design of the primary components of the ClearCausal system.

A. Root Cause Service Inference

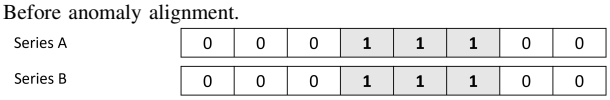
We now describe our causal inference algorithm to localize the root cause services among a large number of inter-dependent microservices. Intuitively, the root cause service often exhibits the causal relationships with the *symptom* service where the performance alert is detected. As aforementioned, we calculate the Granger causality using *mutual information* (MI) to capture both non-linear and linear causality. For variables X_1 and X_2, if previous values of X_1 give information to predict future values of X_2 better than just previous values of X_2, then X_1 "Granger-causes" X_2. MI calculates how much information a variable provides about another. For two random variables X_1 and X_2, the MI formula is outlined in Equation 1.

$$MI(X_1; X_2) = \sum_{x_1 \in X_1} \sum_{x_2 \in X_2} p(x_1, x_2) \log \frac{p(x_1, x_2)}{p(x_1)p(x_2)} \quad (1)$$

where $p(x_1)$ and $p(x_2)$ are the marginal probability mass functions of X_1 and X_2, respectively, and $p(x, y)$ is the joint probability mass function of X and Y.



(a) Before anomaly alignment.



(b) After anomaly alignment.

Fig. 1 An example of the anomaly alignment algorithm. We require a minimum number of consecutive anomalous samples to avoid misalignment due to noise.

We further observe that the anomalies of the root cause service and the performance anomalies are not always perfectly aligned due to clock drift or latent data collection in distributed systems. To accommodate such a challenge, we propose an *anomaly alignment algorithm* to align the anomalies of the failure service and the anomalies of all candidate root cause services before calculating the MI scores. Specifically, after obtaining the anomaly detection result streams, ClearCausal finds the first segment of continuous anomalies in each time series. We then align the start time of the anomaly segments of the two time series to compute a shift length. We then shift the second time series based on the derived shift length to obtain two aligned anomaly time series. For example, Figure 1 shows two anomaly time series before and after alignments, where "1" means the sample is an anomaly, while "0" means it is not an anomaly. We can see that the anomalies from the original time series A and B do not align. After the anomaly continuity check (e.g., at least three continuous anomalies), we identify the start time of the first continuous anomaly segment in time series A is at index 5 and the start time of the first continuous anomaly segment in time series B is at index 6. We can compute the time shift between time series A and time series B is one. So we can align the two time series by shifting the time series B to the left by one.

To localize the root cause in a distributed microservice application consisting of many inter-dependent microservices, we propose to combine the service dependency graph and inter-service causal analysis to achieve high accuracy. To identify the root cause service, we first search the dependency graph from the symptom service to add child nodes at each iteration to the set of root cause service candidates. Next, we compute the causal score between the symptom service and each candidate service. We only consider those candidates whose causal scores are higher than a pre-defined threshold (e.g., 0.5). We then rank all the candidates based on the causal scores. In some rare cases where there are multiple candidates having the same causal scores, we break ties considering whether it is a leaf node (i.e., the leaf nodes rank higher than non leaf nodes since the leaf nodes is not affected by other services, its call count (i.e., higher call counts rank higher since they are more likely have impact), and the anomaly start time (i.e., the earlier anomaly ranks higher).

B. Root Cause Function Identification

To help the developer to fix the performance bugs, we strive to not only localize the root cause service but also the buggy function that triggers the performance problem. ClearCausal does so by analyzing which function contributes to the anomaly of the root cause service. Specifically, we compare the execution time of the functions within the root cause service to the anomalous metrics (e.g., CPU consumption) of the root cause service. We perform causality calculation using a similar algorithm as root cause service localization. In the root cause function analysis, we break ties by prioritizing the function with the earliest change point in execution time.

IV. Experimental Evaluation

In this section, we first describe our evaluation methodology. Next, we compare our system with a set of alternative schemes.

A. Evaluation Methodology

1) Evaluation microservice applications: We evaluate our system on three microservice benchmark applications: Online Boutique [21], Social Network [22], and Media Service [22].

2) Experiment Setup: **Data collection.** We use OpenTelemetry [19] to collect the span duration of the microservices. We use Jaeger [18] to manage the traces, and retrieve the service dependency graph with its internal API. We employ Prometheus [23] to store and query the system metrics of pods, including CPU utilization ratio, active memory usage, network traffic transmitted and received in bytes, and disk read and write in bytes, using non-overlapping moving window of 30 seconds.

Dynamic workload generation. We use Locust [24] to deliver dynamic workloads emulating workloads in real world applications using 10 simulated workers. After a random wait time, each worker performs a random web service task such as visiting the listing pages or adding items to the shopping cart before performing the next task. Each second, Locust records the average response time. Upon detecting the response time change point, we obtain the traces in the window of twelve minute before the change point to two minutes after the change point for analysis.

Anomaly Detection We use the Python sklearn-som package [25] to implement the SOM models of size 32 by 32.

For each performance bug, we repeat the experiment three times and record the average and standard deviation.

B. Alternative Methods

Alternative correlation analysis We evaluated our system with different alternative analysis tools.

- **Pearson correlation** [7]: The Pearson correlation coefficient is a measure of the linear relationship between two continuous variables. Most previous root cause analysis work [8]–[11], [26] adopts the Pearson correlation as the major root cause analysis method.
- **Spearman's rank correlation** [27]: Spearman's rank correlation coefficient is a non-parametric measure of the strength and direction of association between two ranked

TABLE I Summary of the bugs that we use in our experiments.

Bug ID	Symptom	Microservice	Description
1	Infinite loop	OnlineBoutique	The email service encounters an infinite loop when sending the confirmation email.
2	Memory leak	SocialNetwork	GetFollowers in the social graph service does not release the allocated memory.
3	Memory leak	MediaMicroservices	UploadMovieReview in the movie review service does not release the allocated memory.
4	Infinite loop	MediaMicroservices	The user review service falls in an infinite loop when uploading new user reviews.
5	Infinite loop	SocialNetwork	The post storage service encounters an infinite loop when storing new posts.
6	Data corruption (HDFS-5438)	SocialNetwork	The corrupted data stream (text service) affects the loop stride, causing system hanging.
7	Data corruption (Hadoop-8614)	MediaMicroservices	The corrupted data stream (review storage service) returns an error code, causing the system to hang.
8	Timeout (Flume-1819)	SocialNetwork	Timeout is missing in the user mention service, leading to system hanging.
9	Timeout (HDFS-10223)	MediaMicroservices	Timeout is set incorrectly in the rate system, causing system hanging.
10	Timeout (HDFS-4176)	OnlineBoutique	Timeout is missing in the payment service when charging the credit card.

TABLE II Summary of root cause service results.

Bug ID	Number of Candidate Services	Root Cause Service	Rank						False Positives					
			Pearson correlation	Spearman correlation	Kendall correlation	Microscope	MI correlation	ClearCausal	Pearson correlation	Spearman correlation	Kendall correlation	Microscope	MI correlation	ClearCausal
1	47	Email service	5	1	1	2	1	1	4	0	0	1	0	0
2	60	Social graph service	2	11	15	1	10	1	1	10	14	0	9	0
3	66	Movie review service	1	1	1	1	2	1	0	0	0	0	1	0
4	66	User review service	2	5	6	1	24	1	1	4	5	0	23	0
5	60	Post storage service	1	1	1	1	1	1	0	0	0	0	0	0
6	60	Text service	1	10	10	1	1	1	0	9	9	0	0	0
7	66	Review storage service	20	2	3	1	4	1	19	1	2	0	3	0
8	60	User mention service	1	3	3	1	1	1	0	2	2	0	0	0
9	66	Rating service	33	5	5	5	3	1	32	4	4	4	2	0
10	47	Payment service	31	1	1	5	1	1	30	0	0	4	0	0
Average			9.7	4.0	4.6	1.9	4.8	1	8.7	3.0	3.6	0.9	3.8	0

TABLE III Summary of root cause function results. "X" means failure to detect the root cause function.

Bug ID	Number of Candidate Functions	Root Cause Function	Rank						False Positives					
			Pearson correlation	Spearman correlation	Kendall correlation	Microscope	MI correlation	ClearCausal	Pearson correlation	Spearman correlation	Kendall correlation	Microscope	MI correlation	ClearCausal
1	757	SendOrderConfirmation	X	X	X	X	2	1	X	X	X	X	1	0
2	661	GetFollowers	X	X	X	X	X	1	X	X	X	X	X	0
3	644	UploadMovieReview	2	2	2	X	X	1	1	1	1	X	X	0
4	644	UploadUserReview	X	X	X	X	X	1	X	X	X	X	X	0
5	661	StorePost	2	2	2	X	4	2	1	1	1	X	3	1
6	661	ComposeText	3	X	X	X	3	1	2	X	X	X	2	0
7	644	StoreReview	X	X	X	X	X	2	X	X	X	X	X	1
8	661	ComposeUserMentions	2	X	X	X	2	1	1	X	X	X	1	1
9	644	UploadRating	X	X	X	X	X	2	X	X	X	X	X	1
10	757	Charge	X	1	1	X	1	1	X	0	0	X	0	0
Average			X	X	X	X	X	1.3	X	X	X	X	X	0.4

variables. It is based on the ranks of the data rather than the actual values.

- **Kendall rank correlation** [28]: Kendall rank correlation is a non-parametric measure of the strength and direction of association between two ranked variables.
- **Microscope [10]**: Microscope detects the root cause service by using Pearson and service dependency information. However, Microscope cannot perform root cause analysis at the function level.

Evaluation metrics In our evaluation, *Rank* refers to the rank of the actual root cause service or function given by the analysis method in question. We also define the number of false positives (FP) as the services/functions ranked above or tied with the actual root cause service/function.

C. Result Analysis

In this section, we analyze the results of the experiments. We first analyze the results of all ten performance bugs before analyzing the result of the infinite loop case study in the Online Boutique application.

Table I summarizes all ten performance bugs evaluated in the experiment. We evaluate four different types of bugs, including infinite loop, memory leak, data corruption, and timeout. Each type is evaluated in at least two different microservice applications.

Table II depicts the root cause service result summary. Each row in the table includes the rank of the root cause service and the number of false positives for the six algorithms: MI, Pearson, Spearman, Kendall, Microscope, and our ClearCausal algorithm. The tables show that Pearson, Spearman, and Kendall all suffer from a high number of false positives and often fail to rank the root cause service as the first service. MI performs better in most bug cases. However, MI can sometimes produce high false positives (e.g., bug #4) since it is sensitive to data noises. Under those circumstances, we observe that ClearCausal achieves robust analysis by employing anomaly detection and alignment over the raw monitoring data. Although Microscope has better results than those pure correlation methods, it still yields quite a few false alarms. Moreover, Microscope often produces similar correlation probabilities for top ranked services, which makes the developer less confident about the ranking. In contrast, ClearCausal is the only solution that successfully ranks the root cause service as the first one in all of the ten bugs.

Table III shows the summary of the root cause function results. If an algorithm fails to detect the root cause service in the previous step, it is unable to proceed to detect the root cause function in the final step. In this case, we use "X" to indicate the failure of detecting the root cause function. We can see MI fails to detect five bugs, Pearson fails to detect six bugs, and Spearman and Kendall both fail to detect seven bugs.

Microscope does not detect any root cause function since it only supports service level root cause analysis. In contrast, our ClearCausal algorithm can successfully detect the root cause function and rank the root cause functions within the top two in all of the ten cases.

1) Online Boutique application: We now examine the results of the online boutique application (bug #1). For one background service, the recommendation service, we induce a CPU hog at the same time as the root cause injection. This presents a case where another service exhibits coinciding symptoms but is not actually the underlying performance issue. Thus, we expect an effective system to identify the recommendation service as problematic but not as the root cause (email service).

Root Cause Service Analysis We discuss the results of the six tested algorithms using four representative services: the email service (the root cause service), the recommendation service (the CPU hog service), kube-proxy (a Kubernetes component), and the checkout service (a service dependent on the root cause).

The root cause service, email service, ranks the first for two different algorithms: MI and ClearCausal. The results show that those algorithms can successfully identify the root cause service, while the remaining four algorithms, Pearson, Spearman, Kendall, and Microscope fail to do so. Nevertheless, only ClearCausal shows a large difference in correlation value between the email service and the second rank service, which indicates strong confidence in the root cause service.

MI ranks the recommendation service as the second, while Pearson ranks the same service as the first, which meets our high rank expectation of the recommendation service due to the CPU hog symptom. However, because the recommendation service is not ranked as one of the priority candidate services according to subsection III-A, its score is set to 0 in Microscope and ClearCausal.

Kube-proxy, a Kubernetes component, is a network proxy running in each node. With a lot of network traffic during the experiment, we expect this service to be active. Both Spearman and Kendall rank it as the first. However, because this service is not in the same namespace as the Online Boutique, it is reset to 0 in Microscope and ClearCausal.

The checkout service is the caller of the email service. At the end of the checkout process, the checkout service will call the email service to send an confirmation email to the buyer. We expect an intelligent algorithm to not be confused by this service. ClearCausal assigns it a very low correlation value to indicate its low probability of being the root cause service.

Root Cause Function Analysis The root cause function for the infinite loop bug in Online Boutique is SendOrderConfirmation. In the source code of the email service, there are a total of eight functions but only four functions were active during the experiment. We discuss the result of the root cause function analysis after three runs, assuming each algorithm successfully detects the root cause service. We do not include Microscope as it only performs root cause service analysis. With Pearson, Spearman, Kendall, and MI, the root cause function is not listed as the first candidate but second, fourth, fourth, and second, respectively. ClearCausal successfully determines the root cause function as the first candidate. The difference between the first rank and the second rank is also the largest with ClearCausal, which again strengthens our confidence in the result. Here, ClearCausal reports the maximum 1.0 score difference while the other algorithms report differences of less than 0.1.

TABLE IV Overhead measurements of ClearCausal analysis components, given a sampling interval of 30s.

ClearCausal Module	Execution Time (μs)
Service anomaly detection	2.28 $\pm$ 0.03 per sample
Service anomaly alignment	0.12 $\pm$ 0.02 per sample
Root cause service analysis total	31.19 $\pm$ 0.60 per sample
Function anomaly detection	2.24 $\pm$ 0.14 per sample
Function anomaly alignment	0.14 $\pm$ 0.04 per sample
Root cause function analysis total	41.64 $\pm$ 1.09 per sample

TABLE V Overhead measurements of Jaeger. Each experiment is repeated ten times and we report both mean and standard deviation values.

Metric	Overhead (%)
Response Time	2.75 $\pm$ 0.013
CPU utilization	0.25 $\pm$ 0.032

2) Overhead Analysis: ClearCausal is fast and suitable for online analysis as shown in Table IV, and is lightweight as shown in Table V. We observe that ClearCausal incurs less than 1% CPU load to the microservice application and incurs an average response time increase of 2.75% when the tracing is triggered using a sampling rate of 1%. In addition, ClearCausal requires a limited amount of memory to run. During our experiments, the average and maximum memory usage are 123.92 MB and 235 MB, respectively.

V. Related Work

Microservice performance root cause analysis Microscope [10] and MicroRCA [8] use Pearson correlation-based scores as they reason about their dependency graphs. Whereas, ClearCausal shows that applying correlations over raw data directly shows poor accuracy in microservices under dynamic and fluctuating workloads.

Seer [12] processes remote procedure call (RPC) traces with neural architectures to learn conditions for quality of service (QoS) violations. Seer learns from traces annotated with violation labels. Whereas, we do not depend on labeled training data. Sage [14] also operates on RPC-level traces, using causal bayesian networks and unsupervised autoencoders to find culprit services. Sage still relies on historical QoS violation training data and so cannot debug previously unseen QoS violations. In contrast, ClearCausal leverages anomaly detection and causal inference methods to help debug both previously seen or unseen performance problems.

ClearCausal not only provides the root cause service but also provides the root cause function, which is not often localized by the related work. We generally use stronger causality measures with MI than the correlation-based approaches to help us minimize false positives.

Causality analysis PerfSig [6] is a multi-modality performance tool that can identify root cause functions for performance bugs using MI. However, PerfSig cannot provide cross-layer causal analysis or consider service dependency relationships. In contrast, ClearCausal provides cross-layer causal inference, which can localize both root cause services and functions. CauseInfer [29] is a black-box cause inference system that monitors TCP request latency to provide hints for distributed systems bugs. In contrast, ClearCausal leverages MI causal inference and service dependencies to improve the root cause analysis accuracy.

Compared to the causal analysis research, ClearCausal takes a lightweight, non-intrusive approach that finds root causes at the function-level. ClearCausal employs Granger causality with MI to identify dynamic dependency relationships among numerous service nodes.

VI. Conclusion

In this paper, we present ClearCausal, a new dependency-aware cross-layer causal analysis framework to achieve fine-grained performance debugging for microservice applications. ClearCausal combines anomaly detection, information theory based causal analysis, and dependency graphs to pinpoint root cause services and functions when a performance problem is detected. We have developed a prototype of ClearCausal and evaluated it over ten performance bugs in three benchmark microservice applications. The results show that ClearCausal accurately identifies the root cause services for all tested performance anomalies with much fewer false positives compared to traditional correlation methods. Furthermore, ClearCausal can successfully rank the root cause functions within the top two candidates among a large number of functions, which can greatly expedite the debugging process for the developer. ClearCausal is lightweight, which imposes less than 1% CPU overhead and 2.75% performance overhead on average to the micro-service application.

References

[1] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *Ieee Software*, vol. 33, no. 3, pp. 42–52, 2016.

[2] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

[3] A. Kharpal, "Microsoft users hit with global cloud outage that impacted products like teams and outlook," Jan 2023. [Online]. Available: https://www.cnbc.com/2023/01/25/microsoft-investigating-teams-and-outlook-outage-as-users-report-issues.html

[4] J. Mace, R. Roelke, and R. Fonseca, "Pivot tracing: Dynamic causal monitoring for distributed systems," in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 378–393.

[5] S. Kobayashi, K. Otomo, K. Fukuda, and H. Esaki, "Mining causality of network events in log data," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 53–67, 2017.

[6] J. He, Y. Lin, X. Gu, C.-C. M. Yeh, and Z. Zhuang, "Perfsig: Extracting performance bug signatures via multi-modality causal analysis," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1669–1680.

[7] K. Pearson, "Notes on the history of correlation," *Biometrika*, vol. 13, no. 1, pp. 25–45, 1920.

[8] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Microrca: Root cause localization of performance issues in microservices," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[9] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "Automap: Diagnose your microservice-based web applications automatically," in *Proceedings of The Web Conference 2020*, 2020, pp. 246–258.

[10] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 3–20.

[11] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia, Z. Wang, and D. Pei, "Localizing failure root causes in a microservice through causality inference," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 2020, pp. 1–10.

[12] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou, "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*, 2019, pp. 19–33.

[13] Q. Wang, L. Shwartz, G. Y. Grabarnik, V. Arya, and K. Shanmugam, "Detecting causal structure on cloud application microservices using granger causality models," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 558–565.

[14] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: practical and scalable ml-driven performance debugging in microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 135–151.

[15] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "Prepare: Predictive performance anomaly prevention for virtualized cloud systems," in *2012 IEEE 32nd International Conference on Distributed Computing Systems*. IEEE, 2012, pp. 285–294.

[16] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *Proceedings of the 9th international conference on Autonomic computing*, 2012, pp. 191–200.

[17] J. T. Lizier, "Jidt: An information-theoretic toolkit for studying the dynamics of complex systems," *Frontiers in Robotics and AI*, vol. 1, p. 11, 2014.

[18] T. J. Authors, "Jaeger: open source, end-to-end distributed tracing," 2023. [Online]. Available: https://www.jaegertracing.io/

[19] T. O. Authors, "Opentelemetry," 2023. [Online]. Available: https://opentelemetry.io/

[20] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1-3, pp. 1–6, 1998.

[21] O. B. Authors, "Googlecloudplatform/microservices-demo: Sample cloud-first application with 10 microservices showcasing kubernetes, istio, and grpc." 2023. [Online]. Available: https://github.com/GoogleCloudPlatform/microservices-demo

[22] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 3–18.

[23] P. Authors, "Prometheus - monitoring system & time series database," 2023. [Online]. Available: https://prometheus.io/

[24] L. Authors, "Locust - a modern load testing framework," 2023. [Online]. Available: https://locust.io/

[25] "Sklearn-som," 2023. [Online]. Available: https://pypi.org/project/sklearn-som/

[26] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 93–104, 2013.

[27] C. Spearman, "The proof and measurement of association between two things." 1961.

[28] H. Abdi, "The kendall rank correlation coefficient," *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA*, pp. 508–510, 2007.

[29] P. Chen, Y. Qi, P. Zheng, and D. Hou, "Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 1887–1895.