# MicroDiag: Fine-grained Performance Diagnosis for Microservice Systems

Li Wu*†, Johan Tordsson*‡, Jasmin Bogatinovski†, Erik Elmroth*‡, Odej Kao†
*Elastisys AB, Umeå, Sweden, Email: {li.wu, johan.tordsson, erik.elmroth}@elastisys.com
†Distributed and Operating Systems Group, TU Berlin, Berlin, Germany, Email: jasmin.bogatinovski, odej.kao@tu-berlin.de
‡Department of Computing Science, Umeå University, Umeå, Sweden

***Abstract*—Microservice architecture has emerged as a popular pattern for developing large-scale applications for its benefits of flexibility, scalability, and agility. However, the large number of services and complex dependencies make it difficult and time-consuming to diagnose performance issues. We propose MicroDiag, an automated system to localize root causes of performance issues in microservice systems at a fine granularity, including not only locating the faulty component but also discovering detailed information for its abnormality. MicroDiag constructs a component dependency graph and performs causal inference on diverse anomaly symptoms to derive a metrics causality graph, which is used to infer root causes. Our experimental evaluation on a microservice benchmark running in a Kubernetes cluster shows that MicroDiag localizes root causes well, with 97% precision of the top 3 most likely root causes, outperforming state-of-the-art methods by at least 31.1%.**

***Index Terms*—Performance diagnosis, Microservice system, Causal inference, Data-driven, Fine-grained root cause.**

## I. Introduction

Microservices architecture (MSA) has been adopted in many domains such as edge computing, internet of things (IoT), and cloud computing [1], to design large-scale applications because of its benefits of resilience, scalability, and acceleration of software delivery, etc [2]. With MSA, an application is decomposed into autonomous units that can be deployed independently and intercommunicate with lightweight protocols. However, due to the highly distributed and dependent nature of microservice systems, performance issues are inevitable and it is infeasible for operators to diagnose them manually [3].

To resolve performance issues quickly, it is critical to automate the diagnosis process and pinpoint root causes at a fine granularity, i.e., identifying both the faulty component and details about why it is faulty (see Section III for a formal problem formulation). However, identifying fine-grained root causes in a microservice system is extremely difficult because of the following challenges. *1) A large number of anomalous metrics*: A performance issue introduces anomalies to one or more metrics initially, but tends to propagate quickly to also create anomalies in other metrics, unrelated to root cause. It is a challenging task to pinpoint root causes from a large number of anomalous metrics; *2) Heterogeneous anomaly symptoms*: Microservices can be polyglot and differ in other characteristics as well, which might lead to diverse anomaly symptoms for the same issue. *3) Frequently updates:* Microservices are frequently updated to meet customers' needs (e.g., Netflix updates thousands of times per day [4]), which might make the historical failure data obsolete. Hence, the performance diagnosis system must be able to adapt to changes. *4) A wide range of causes:* Due to the complexity of microservice systems, the root cause for a performance issue can vary significantly, but issues can be broadly categorized as internal to a specific microservice (software bugs, configuration issues, etc) or external (third-party database, spike workload, etc). These causes have various symptoms, making it hard to localize them from observable metrics in microservice systems.

In the literature (Section II), various approaches based on deep learning [5], [6], pattern recognition [7], and causal inference [8], [9] have been proposed to identify fine-grained root causes for performance issues. However, deep learning based methods requires frequently retraining to follow up the updates in microservices and pattern recognition based methods have a high computation complexity and can only identify root causes for known issues. The third approach based on causal inference techniques constructs causality graph with metrics from multiple components and may fail to identify root causes that have diverse anomaly symptoms.

In this paper, we propose an application-agnostic system named MicroDiag (Section IV) for real-time performance diagnosis in microservice systems without requiring any historical anomaly data. MicroDiag continuously collects metrics from components and detects anomalies on SLO (Service Level Objective) metrics. Once an anomaly is detected, MicroDiag infers fine-grained root causes by modeling the anomaly propagation with a metrics causality graph and ranking the culprit metrics by traversing along this graph. The metrics causality graph is derived from a component dependency graph with two causal inference methods, which are employed to detect anomaly propagation paths from diverse anomaly symptoms. As the metrics causality graph is based on causal inference between inter-dependent components, MicroDiag can easily scale to the number of components in the system.

We evaluate MicroDiag on a microservice system running on Google Cloud Engine[1] where the Sock-shop[2] microservice benchmark is deployed using Kubernetes and anomalies CPU hog and memory leaks are injected to different microservices (Section V). The evaluation show that our system can identify

[1]Google Cloud Engine - https://cloud.google.com/compute/
[2]Sock-shop - https://microservices-demo.github.io/

97% of all root causes in one of the top 3 most likely causes, which outperforms the state-of-the-art methods by at least 31.1%.

## II. Related Work

In the literature, different approaches have been proposed to diagnose performance issues in cloud computing [10], computer networks [11], and microservices [12], [13]. Overall, these approaches employ machine learning [5], pattern recognition [7], and causal inference [9] to infer root causes. Some approaches aim to identify the faulty service or server node that initiates performance issues based on different observational data [12]–[14], such as tracing data, logs, and metrics. Here, we only discuss approaches for fine-grained performance diagnosis, which locates not only the faulty components but also hints for recovery.

Machine learning approaches [5], [6] diagnose root causes by identifying features that significantly deviate from the predicted values estimated by machine learning algorithms. This kind of approach is based on the assumption that the anomalous causal feature has a significant deviation from its normal behavior. For example, reconstruction error from autoencoder is used to rank root causes in [6]. Due to this assumption, machine learning approaches are limited to diagnose issues whose anomalous features can manifest significant deviations from their normal status. Besides, due to frequent updates of microservices, retraining might be frequently required, which is time-consuming and inefficient.

Álvaro Brandón, et al. [7] define the performance diagnosis as a pattern recognition problem. They first label and store anomaly graphs from previous failures, then identify the root cause of a new anomaly by matching the anomalous pattern with previous knowledge. This method can identify the root cause at a fine-granularity if the anomaly graphs are labeled with fine-grained root causes. However, as the anomalous patterns are based only on previous anomalies, the method is limited to known issues. In addition, the computation complexity of graph matching is exponential to the size of the stored anomalous patterns.

Different causal inference methods has been applied to learn the causal relations among metrics. The Sieve [15] and Loud [8] systems construct anomaly propagation graph across all metrics using the Granger causality test. However, this test assumes time-lag between cause-effect metrics, which is quite restrictive for some observational metrics (e.g., resource metrics) in microservice systems. In MicroDiag, Granger causality tests are used only for accumulative effects and a structural equation model to infer the causal relations among contemporary metrics. Furthermore, MicroDiag limits the causal inference between inter-dependent components, which can eliminate a large number of spurious causal-effect relationships.

CauseInfer [9] infers the root cause by constructing a metrics causality graph for each service with PC algorithm ((named after its authors, Peter and Clark [16]), and each service is connected to other services by a service dependency graph. It gets the candidate culprit metrics by traversing the metrics causality graph of each service and ranks them with the significance of abnormality. However, with CauseInfer, it is difficult to know which service should be traversed first, and the method is sensitive to the root cause ranking results of individual service. In addition, instantaneous effects among resource metrics are also difficult to identify. MicroDiag constructs a metrics causality graph with metrics from all components, furthermore, the causality graph is constructed with the consideration of diverse anomaly propagation patterns in different components, which yields a higher accuracy in identifying causal-effect relationships and also root causes (see evaluation in Section V).

A variant of the PC algorithm that considers time-order of metrics is used in MicroCause [17] to identify causality graph of metrics from multiple layers in microservice systems. However, as the time-series PC algorithm assumes time-lag between metrics, and conducts conditional independence test for data points of a metric, it would introduce high computation overhead and still cannot identify comtemporary effects well. MicroDiag differentiates the causal discovery of metrics from different layers, which is computationally efficient and scalable to the number of components. Additionally, MicroDiag constructs a component dependency graph which can largely reduce spurious causal-effects, therefore improving the performance of root cause localization.

## III. Problem Formulation

In a typical microservice system, services run inside containers deployed in (virtual or physical) servers that are part of a cluster. We define services, containers, and servers as components $C$. In order to timely detect unexpected behaviors and assist the diagnosis, microservice systems are always equipped with a telemetry infrastructure, which provides extensive metrics time series $M$ for components. We denote $M^c$ as metrics exposed by component $c \in C$, and $m_i^c$ as an individual metric (e.g., response time, CPU utilization, read time of disk IO) of component $c$, where $i$ is the type of metrics.

Based on above definition, the fine-grained performance diagnosis problem is formulated as follows: Given a set of components $C$ and their exposed metrics $M$ in a microservice system, assuming one or more anomalies are detected on response times $m_{rt}$ of microservices, how can we identify the culprit metric $m_{rc}^{c_{rc}}$ that initiates the performance issue? A culprit metric $m_{rc}^{c_{rc}}$ indicates not only the faulty component $c_{rc}$ but also the type of metrics $m_{rc}$ that causes the abnormality of $c_{rc}$. For example, the culprit metric $m_{cpu\_utilization}^{s_1}$ indicates that a high CPU utilization of service $s_1$ causes the performance degradation.

## IV. System Design

In this section, we introduce details of the design of our system MicroDiag for fine-grained performance diagnosis in microservice systems.

### A. System Overview

To address the fine-grained performance diagnosis problem, we propose a system named MicroDiag, which can automatically localize root causes from observable metrics in real time without any application instrumentation.

Figure 1 shows the overview of MicroDiag. Overall, there are five steps in MicroDiag to identify the fine-grained root cause. First of all, it continuously collects metrics data from microservice system and groups them by components as $M^c$ after preprocessing. Meanwhile, the response times of all microservices are monitored by the anomaly detection module. Once anomalies (unusually slow response times) are detected, the root cause localization process is triggered. In this process, MicroDiag firstconstructs a component dependency graph $DG$ to model anomaly propagation across components, then infers causal relations among metrics of inter-dependent components using causal inference techniques and outputs a metrics causality graph $MG$. Finally, MicroDiag weighs the metrics causality graph and ranks culprit metrics $m_{rc}^{c_{rc}}$ with a graph centrality algorithm, where the highest ranked metric has the highest probability to be the root cause.

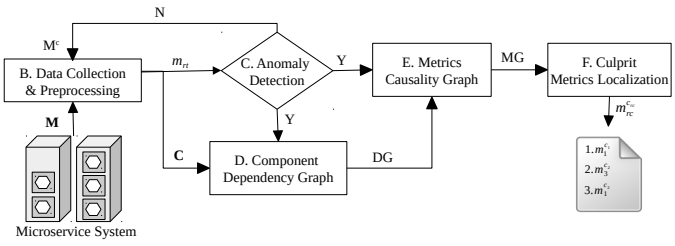


Figure. 1. Overview of MicroDiag.

### B. Data Collection and Preprocessing

MicroDiag is designed to be application-agnostic. It collects metrics exposed by components from multiple layers, including microservices, containers and server nodes, without any instrumentation. It adopts a cloud-native monitoring stack, where Node-exporter[3] is used to monitor operating system of server nodes; Cadvisor[4] to monitor resources of containers and service mesh Istio[5] to monitor the network interactions between microservices, including response times. Once metrics are collected, the unvarying metrics will be removed and the rest of metrics will be grouped by components.

### C. Anomaly Detection

MicroDiag detects anomaly on the response times $m_{rt}$ of microservices using an unsupervised learning algorithm named Distance-Based online clustering BIRCH. For each response time which is a time series, MicroDiag applies the Birch clustering and detects it as an anomaly if multiple clusters are detected with a given threshold. Birch clustering is an efficient algorithm for anomaly detection which detects anomalies in real-time without relying on any historical failure data [18].

[3]Node-exporter - https://github.com/prometheus/node_exporter
[4]Cadvisor - https://github.com/google/cadvisor
[5]Istio - https://istio.io/

### D. Component Dependency Graph Construction

In order to infer the anomaly propagation across metrics, MicroDiag firstly constructs a component dependent graph to show the potential propagation across components, then detects the anomaly propagation across metrics of dependent components. By limiting the causal inference among metrics of inter-dependent components, the method can largely reduce spurious causal relations between unrelated metrics.

In a microservice system (e.g., Figure 2(a)), an anomaly can propagates across not only inter-dependent microservices along invocations but also containers and severs through resources sharing or contention (such as $s_1$ and $s_3$ in Figure 2(a)). To capture such dependencies, we construct a component dependent graph among microservices, containers, and servers to model the potential anomaly propagation paths.

For each service, we add edges to all other services it communicates with and all containers it runs inside; For a container, we add edge to service it runs and server it runs on. For a server, we add edges to all containers it runs. Thus, we get a directed graph where some edges are bi-directed. We discover the graph nodes and their dynamic relationships by enumerating and parsing the metrics exposed by the components, which is an extension of the attributed graph proposed in our previous work on root cause analysis for microservices [13]. Specifically, we parse the service-level metrics collected by service mesh istio[5] to get the service dependencies and the system metrics collected by Cadvisor[4] to get the deployment information. Figure 2(b) gives an example of the component dependency graph of the microservice system in Figure 2(a), including service dependencies and deployment relationships among services, containers, and servers.

### E. Metrics Causality Graph Inference

After obtaining the component dependency graph, MicroDiag use causal inference methods to construct a metrics causality graph across metrics of inter-dependent components.

In the metrics causality graph, each node represents an individual metric $m_i^c$ of a component $c$ and an edge represents a causal-effect relationship which is also an anomaly propagation path. For example, an edge from metric $m_1^{p_3}$ to metric $m_2^{p_3}$ in Figure 2(c), means $m_1^{p_3}$ causes $m_2^{p_3}$ and the anomaly propagates from $m_1^{p_3}$ to $m_2^{p_3}$.

In order to identify the causal relations among metrics, MicroDiag divides the anomaly propagation properties into three types: i) propagation across resource metrics; ii) propagation across resource and service metrics; iii) propagation across services.

Regarding i) anomaly propagation across resource metrics, MicroDiag employs a structural causal model (SCM) [19] to infer the causality. An anomaly from one resource simultaneously propagates to other resources, resulting in contemporary effects, which are hard to identify through conditional independence tests, like the PC algorithm, and time-lag based methods used in the existing methods [8], [9] but can be handled by SCM methods. SCM methods assume that the value of each

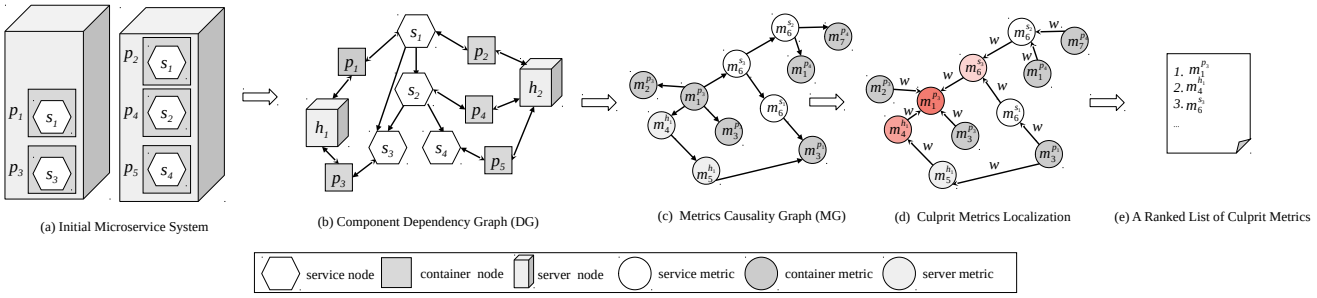


Figure. 2. Root cause localization procedures in MicroDiag.

variable is a deterministic function of its direct causes and some independent factors, like measurement errors. Notably, server resources are sums of the containers resources that it runs, i.e., a linear function. It is also assumed that anomalous resource metrics follows a non-gaussian distribution [20]. Therefore, we employ a non-gaussian linear model to infer the anomaly propagation paths across resource metrics in dependent containers and servers. A metric $m_i^{c_x}$, is defined as a linear function of causal metrics $m_j^{c_y}$ that the anomaly propagates from, and an independent factor $e_i^{c_x}$, where the latter from the measurement errors or the fluctuations of metrics. Formally, $m_i^{c_x}$ can be defined as Equation 1. MicroDiag employs a method named DirectLiNGAM [21] to infer the causality from the non-gaussian linear model. It first identifies an exogenous variable based on its independence of the residuals of a number of pairwise regression, then removes the effect of the exogenous variable from the other variables using least squares regression. Based on these iterations of effect removal and causal ordering, it returns the causal relations of all metrics. The causality among resource metrics is shown as the container and server metrics in Figure 2(c).

$$m_i^{c_x} = \sum_{k(j)<k(i)} b_{ij} m_j^{c_y} + e_i^{c_x} \tag{1}$$

For ii) anomaly propagation across resource and service metrics, MicroDiag leverages Granger causality tests to infer the causality as the anomaly propagates in an accumulative way, where a cause precedes an effect. Granger Causality tests are useful to determine whether a time series can be used to predict another time series and is defined as follows: A time series $X$ is said to *Granger-cause* another time series Y if including information about the past of X significantly increases the prediction accuracy of the current value of Y in comparison to predicting future Y based on past values of Y.

For Granger causality tests, MicroDiag first gets the time-lag between two metrics based on Akaike information criterion (AIC), then infers causal relation by applying Granger causality with $\chi^2$ test. As Granger causality tests might introduce spurious causal relations among container resource metrics, we calibrate the causality with the results from above SCM model. The causality among resource and service metrics is shown as the container and service metrics in Figure 2(c).

Once MicroDiag identifies the causality of each service with above two steps, it further infers iii) the anomaly propagation across services, by reversing the edges between services in component dependent graph. In the end, MicroDiag gets a metrics causality graph of all metrics. Figure 2(c) gives an example of the constructed metrics causality graph from component dependency graph Figure 2(b).

### F. Culprit Metrics Localization

Once metrics causality graph is obtained, MicroDiag infers culprit metrics along the graph with a graph centrality algorithm which is commonly used in the state-of-the-art [8], [13].

MicroDiag first weigh the graph with pearson correlation coefficient between two metrics to show the probability of anomaly propagation, then ranks the culprit metrics with PageRank. The transition probability matrix $\boldsymbol{P}$ in PageRank is computed as $P_{ij} = \frac{w_{ij}}{\sum_j w_{ij}}$ if node $i$ links to node $j$, and $P_{ij} = 0$ otherwise, and the teleportation probability is set to $c = 0.15$ as recommended in [22]. By ranking the nodes in the metrics causality graph, MicroDiags returns a ranked list of potential root causes. We note that the metrics causality graph needs to be reversed before conducting the localization procedure, as shown in Figure 2(d). An example of the ranked list of culprit metrics is shown in Figure 2(e), where $m_1^{p_3}$ has the highest probability to be the root cause.

## V. Experimental Evaluation

In this section, we set up a testbed to evaluate the effectiveness of MicroDiag on locating culprit metrics from anomaly cases, and also compare our method to two baseline methods.

### A. Experimental Setup

**Experimental Testbed:** We develop a prototype of our system and evaluate it in a testbed created in Google Cloud Engine[1] using Kubernetes, where we deploy a microservice benchmark named Sock-shop[2], which has 13 microservices, and a set of cloud-native monitoring tools, including Istio[5], Cadvisor[4], Node-exporter[3] and Prometheus[6]. In our cluster, there are one master node and four worker nodes (4 vCPU and 15 GB) with container-Optimized OS, including three worker nodes dedicated for microservices and one for data collection. Besides, we developed a load generator based on Locust[7] and run it on a virtual machine (6 vCPU and 12 GB) outside of the cluster.

[6]Prometheus - https://prometheus.io/
[7]Locust - https://locust.io/

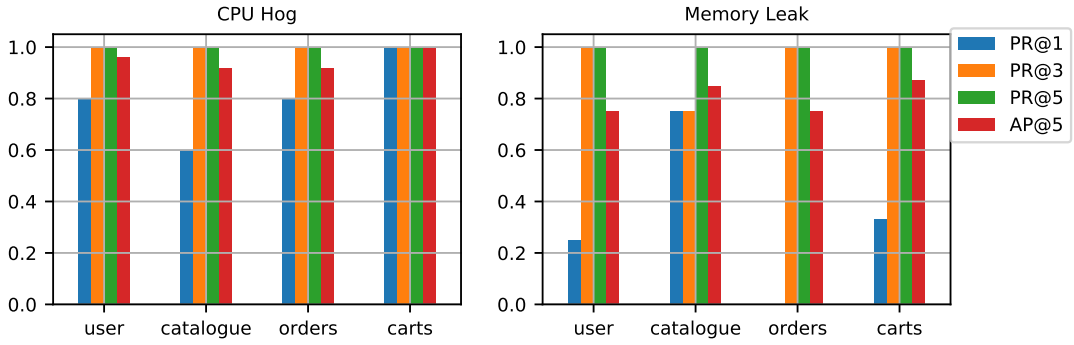


Figure. 3. Performance of MicroDiag in terms of PR@1, PR@3, PR@5, and AP@5.

**Fault Injection:** We inject two types of performance issues: CPU hog and memory leak, by exhausting resource CPU and memory in a container, with stress-ng[8]. These two types of faults are commonly used in the state-of-the-art [8], [9] , and can be manifested in the monitoring metrics. In order to simulate the performance issues in containers that microservices run inside, we customize the Docker images of each microservice by installing the fault injection tools. For each anomaly case, we run the system in normal status for 5 minutes, then we inject one anomaly to one of four microservices (catalogue, carts, orders, and users) for 1 minutes and wait for another 5 minutes for cold down in the system before the next fault injection. We repeat our experiments for 5 times and in total we have 40 cases.

**Baseline Methods:** We compare our system to two state-of-the-art methods as follows:

- Loud [8]: Loud localizes the culprit metrics by constructing a propagation graph of anomalous metrics using the Granger causality test. To implement Loud, we use the anomalous metrics detected by our anomaly detection and construct the propagation graph using the Granger causality test, then rank the culprit metrics using PageRank after assigning weights to edges.
- CauseInfer [9]: CauseInfer identifies the culprit metrics by constructing a service dependency graph and metrics causality graphs for each service using the PC algorithm. To implement CauseInfer, we use the dependency among services in our component dependency graph as the service dependency graph, then use PC algorithm with partial correlation to get the metrics causality graph, and rank the culprit metrics with our localization method. We note that the independent test and ranking method in CauseInfer have poor performance in our dataset.

**Evaluation Metrics:** To quantify the performance of each system, we use the following two performance metrics:

- *Precision at top k* denotes the probability that the top $k$ results given by a system include the real root cause, denoted as $PR@k$. A higher $PR@k$ score, especially for small values of $k$, represents the system correctly identifies the root cause, Let $R[i]$ be the rank of each cause and $v_{rc}$ be the set of root causes. More formally, $PR@k$ is defined on a set of given anomalies $A$ as:

$$PR@k = \frac{1}{|A|} \sum_{a \in A} \frac{\sum_{i<k}(R[i] \in v_{rc})}{(min(k, |v_{rc}|))} \tag{2}$$

- *Average Precision at k* (AP@k) quantifies the overall performance of a system with an average of PR@k:

$$AP@k = \frac{1}{k} \sum_{1 \le j \le k} PR@j. \tag{3}$$

[8]stress-ng - https://kernel.ubuntu.com/ cking/stress-ng/

### B. Effectiveness Evaluation

Figure 3 shows the performance of MicroDiag in identifying root cause from two types of anomalies injected to four microservices. Overall, we can see that MicroDiag achieves a higher performance in CPU hog than memory leak in PR@1, with 80% in PR@1 of CPU hog and 33% of PR@1 for memory leak. This is because i) memory leak issue manifests in multiple resource metrics, which is more difficult to identify causal-effect relationships; ii) The monitoring interval is 5 seconds and it might fail to capture the changes in time, thus fails causal inference. Besides, we can see that MicroDiag pinpoints root cause well as one of top 3 mostly likely causes, with an average of 100% and 93% for CPU hog and memory leak respectively.

### C. Comparison

Furthermore, we evaluate the performance of MicroDiag by comparing with two baseline methods (Loud and CauseInfer).

We apply MicroDiag and two baseline methods to all anomaly cases and get the average performance of each method in terms of PR@1, PR@3, PR@5 and AP@5, as shown in Table I. We can see that all these methods cannot pinpoint the culprit metric in top 1 of the ranked list. However, comparing to the baseline methods, our MicroDiag achieves a large improvement with at least 76.5% in precision, and it achieves 97% in PR@3 and 100% in PR@5. Besides, we can see that CauseInfer has a poor performance in identifying root causes, particularly, with 9% in PR@1. This is because PC algorithm has a low performance of discovering contemporary effects resource metrics, thus failing to pinpoint root causes well. In opposite, Loud uses Granger causality tests which adds spurious causal relations among metrics. With the aids

TABLE I
PERFORMANCE OF EACH SYSTEM.

| Metric | Loud | CauseInfer | MicroDiag | Minimum improvement to baseline methods (%) |
|---|---|---|---|---|
| PR@1 | 34 | 9 | 60 | 76.5 |
| PR@3 | 74 | 49 | 97 | 31.1 |
| PR@5 | 94 | 69 | 100 | 6.4 |
| AP@5 | 70 | 42 | 89 | 27.1 |

of PageRank, it has a high chance to identify the root cause, therefore, it achieves a higher performance in PR@5.

## VI. CONCLUSIONS

In this paper, we propose an automated performance diagnosis system named MicroDiag for microservice systems. MicroDiag infers fine-grained root causes from metrics, which indicates not only faulty component but also the clues for the abnormality of faulty component, by modeling the anomaly propagation across metrics with a causality graph. To achieve the metric causality graph, MicroDiag identifies the potential propagation across components firstly, then infers the causal relations among metrics of inter-dependent components with causal inference techniques. As the property of anomaly propagation across metrics are different, two types of causal inference methods are used. We evaluate our system on a testbed running on google cloud engine where a microservice benchmark is deployed on a Kubernetes cluster and two types of performance issues are injected. The experimental results show that MicroDiag can rank 97% of the culprit metrics in one of the top 3 most likely causes, and outperforms at least 31.1% of the state-of-the-art methods.

So far, we only evaluated our system on two types of faults. In the future, we will extend our evaluation on more types of faults and also a large microservice benchmark, like TrainTicket. Besides, we would like to improve our MicroDiag with feature selection to reduce the dimensionality of metrics.

## ACKNOWLEDGMENT

This work is part of the FogGuru project which has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 765452. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

## REFERENCES

[1] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 21–30.

[2] S. Newman, *Building Microservices*. O'Reilly Media, Inc, USA, 2015.

[3] P. Jamshidi *et al.*, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

[4] *Why Netflix, Amazon, and Apple Care About Microservices*. [Online]. Available: https://blog.leanix.net/en/why-netflix-amazon-and-apple-care-about-microservices, (accessed: 01.18.2021).

[5] 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 *ASPLOS '19*, 2019, pp. 19–33.

[6] L. Wu, J. Bogatinovski, S. Nedelkoski, J. Tordsson, and O. Kao, "Performance Diagnosis in Cloud Microservices using Deep Learning," in *AIOPS 2020 - International Workshop on Artificial Intelligence for IT Operations*, 2020.

[7] Á. Brandón *et al.*, "Graph-based root cause analysis for service-oriented and microservice architectures," *Journal of Systems and Software*, vol. 159, p. 110 432, 2020.

[8] L. Mariani, C. Monni, M. Pezzé, O. Riganelli, and R. Xin, "Localizing faults in cloud systems," in *ICST*, 2018, pp. 262–273.

[9] 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*, 2014, pp. 1887–1895.

[10] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, pp. 1–35, 2015.

[11] M. łgorzata Steinder and A. S. Sethi, "A survey of fault localization techniques in computer networks," *Science of computer programming*, vol. 53, no. 2, pp. 165–194, 2004.

[12] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *ESEC/FSE 2019*, 2019, pp. 683–694.

[13] 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.

[14] P. Zhou *et al.*, "Logsayer: Log pattern-driven cloud component anomaly diagnosis with machine learning," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 2020, pp. 1–10.

[15] J. Thalheim, A. Rodrigues, I. E. Akkus, P. Bhatotia, R. Chen, B. Viswanath, L. Jiao, and C. Fetzer, "Sieve: Actionable insights from monitored metrics in distributed systems," in *Middleware '17*, 2017, pp. 14–27.

[16] P. Spirtes, C. N. Glymour, R. Scheines, and D. Heckerman, *Causation, prediction, and search*. MIT press, 2000.

[17] Y. Meng *et al.*.

[18] A. Gulenko, F. Schmidt, A. Acker, M. Wallschläger, O. Kao, and F. Liu, "Detecting anomalous behavior of black-box services modeled with distance-based online clustering," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 912–915.

[19] J. Pearl *et al.*, "Causal inference in statistics: An overview," *Statistics surveys*, vol. 3, pp. 96–146, 2009.

[20] C. Loboz, "Cloud resource usage—heavy tailed distributions invalidating traditional capacity planning models," *Journal of grid computing*, vol. 10, no. 1, pp. 85–108, 2012.

[21] S. Shimizu, T. Inazumi, Y. Sogawa, A. Hyvärinen, Y. Kawahara, T. Washio, P. O. Hoyer, and K. Bollen, "Directlingam: A direct method for learning a linear non-gaussian structural equation model," *The Journal of Machine Learning Research*, vol. 12, pp. 1225–1248, 2011.

[22] G. Jeh and J. Widom, "Scaling personalized web search," in *WWW '03*, 2003, pp. 271–279.