Causal Inference Techniques for Microservice Performance Diagnosis: Evaluation and Guiding Recommendations

Li Wu, Johan Tordsson, Erik Elmroth, Odej Kao

Causal Inference Techniques for Microservice Performance Diagnosis: Evaluation and Guiding Recommendations

Li Wu*†, Johan Tordsson*‡, 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: odej.kao@tu-berlin.de
‡Department of Computing Science, Umeå University, Umeå, Sweden

***Abstract*—Causal inference (CI) is one of the popular performance diagnosis methods, which infers the anomaly propagation from the observed data for locating the root causes. Although some specific CI methods have been employed in the literature, the overall performance of this class of methods on microservice performance diagnosis is not well understood. To this end, we select six representative CI methods from three categories and evaluate their performance against the challenges of microservice operations, including the large-scale observable data, heterogeneous anomaly symptoms, and a wide range of root causes. Our experimental results show that 1) CI techniques must be integrated with anomaly detection or anomaly scores to differentiate the causality in normal and abnormal data; 2) CI techniques are more robust to false positives in anomaly detection than knowledge-based non-CI method; 3) To get the fine-grained root causes, an effective way with CI techniques is to identify the faulty service first and infer the detailed explanation of the service abnormality. Overall, this work broadens the understanding of how CI methods perform on microservice performance diagnosis and provides recommendations for an efficient application of CI methods.**

***Index Terms*—Performance diagnosis, Microservices, Causal inference, Experimental evaluation, Self-healing**

I. Introduction

Microservice architecture is a popular paradigm for designing large-scale applications because of its benefit of accelerated delivery time, resilience to failures, and flexible development and deployment [1]. A microservice-based application is composed of loosely-coupled services, which can be developed with heterogeneous technology stacks and communicate through lightweight protocols [2]. However, performance issues are inevitable in mcroservice due to its large-scale services, complex dependencies, and frequent updates.

With the aid of monitoring tools and anomaly detection in microservices, a large number of measurements can be observed and unexpected behaviors can be detected. However, automatic diagnosis of root causes from the observed data for is difficult for the following challenges. *1) A large volume of anomalous metrics:* a performance issue is likely to result in a large number of components emitting anomalous metrics, as the anomaly tends to propagate horizontally across inter-dependent services or resources as well as vertically across multiple layers. To illustrate the phenomena, Figure 1 shows that metrics from multiple data resources (microservice

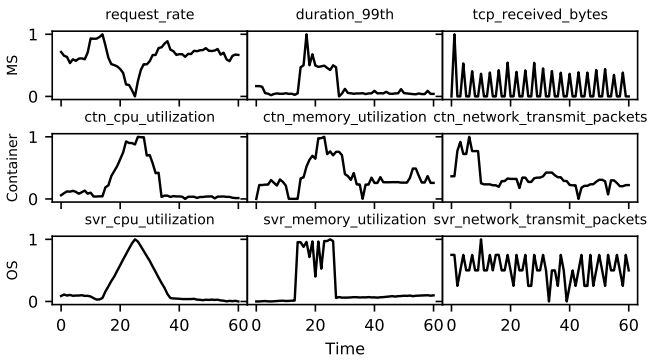


Figure 1: Multiple metrics become abnormal due to the anomaly propagation from a CPU hog issue. Moreover, anomaly patterns are different in metrics due to the heterogeneity and dynamics in microservices and also the control mechanisms in the system.

(MS), container, and operating system (OS)) become abnormal when a service exhausts the allocated CPU resource. To complicate further, the number of services can be very large in microservices (e.g., Uber reports 4000 services deployed [3]), amplifying the volume of anomalous metrics. *2) Heterogeneous anomaly symptoms:* a performance issue tends to manifest heterogeneous anomaly patterns, due to different types of metrics, frequently updated and polyglot services, and the control mechanisms in the system (e.g., load-balancing, auto-scaling, restart, etc). As shown in Figure 1, a CPU hog issue has different anomaly patterns in the metrics of duration, request rate, and resources. *3) A wide range of root causes:* root causes of a service performance degradation can be external (e.g., hardware failures) or internal issues (e.g., software bugs), and the causes keep evolving with the frequent updates of microservices. Therefore, historical failure data is unable to contain instances of all root causes, failing the historical data based diagnosis methods.

A variety of approaches have been proposed in the literature to locate performance issues in microservice systems (briefly discussed in Section V). One of the dominant approaches is to formulate performance diagnosis as a causal inference problem, which constructs an anomaly propagation graph based on causal inference for locating the root causes. For

example, Microscope [3] applies the PC algorithm (named after its authors, Peter and Clark) and Loud [4] uses Granger causality to obtain the graphs. However, only few CI methods are studied in the existing approaches, and their performance against the challenges lying in microservice performance diagnosis is not well understood.

To this end, we select six representative CI methods from three categories including Granger causality, causal network learning algorithms, and structural equation models (SCM), and evaluate their performance on locating root causes against several dimensions of the challenges in microservice performance diagnosis (the background of CI techniques is introduced in Section II). The root cause diagnosis is based on observable metrics collected from a set of fault injection experiments, where three types of anomalies are injected into multiple different services of two widely used microservice benchmarks. For comparison, we developed a non-CI method that leverages the domain-knowledge of service dependencies. To the best of our knowledge, this work is the first such comprehensive study to be reported on evaluating CI techniques for microservice performance diagnosis. The empirical results presented in this work are useful to understand how well CI methods perform for root cause analysis so as to make a proper decision.

To sum up, our contributions are following threefold:

1) A method for evaluating the performance of CI techniques on microservice performance diagnosis. In this method, we formulate three research questions and implement multiple anomaly scenarios to showcase several dimensions of the challenges, including the large number of metrics, heterogeneous anomaly patterns, and a wide range of root causes (Section III).
2) A comprehensive comparison of six representative CI methods on microservice performance diagnosis, including the accuracy of coarse-grained and fine-grained causes localization, computation overhead, and scalability. We also compare CI methods against a non-CI method based on domain-knowledge (Section IV).
3) Our evaluation shows that CI methods need to incorporate anomaly symptoms for performance diagnosis, and the heterogeneity of anomaly symptoms decreases the performance of CI methods. Based on the observations, several research directions are proposed (Section VI). In conclusion, this work not only deepens the understanding of CI methods against the challenges lying in microservice performance diagnosis but also enables new research directions, like performance diagnosis with SCM models.

II. Background

A. Coarse-grained and fine-grained diagnosis

Given a collection of services S and their performance metrics M in a microservice system, we use $M^{(s)}$ to denote metrics for service s, and $m_i^{(s)}$ for an individual metric (e.g., duration, container CPU utilization, server filesystem utilization, etc) for service s. Then, a performance diagnosis problem can be formulated as: giving a set of k metrics $M = \{m_1^{(s_1)}, m_2^{(s_1)}, m_1^{(s_2)}, ..., m_k^{(s_n)}\}$ in a system with n services, once a service performance degradation is detected, how can we pinpoint the root cause $m_{rc}^{(s_{rc})}$ that first identifies the anomaly? We define a *coarse-grained diagnosis* as when metrics are from service-level (e.g., duration) and only faulty service s_{rc} can be identified, and a *fine-grained diagnosis* as when metrics from multiple data resources (e.g., service, container, and server) and the detailed explanation of service's abnormality $m_{rc}^{(s_{rc})}$ can be identified.

The main objective of applying CI methods to performance diagnosis is to identify the causal graph with giving (anomalous) metrics, showing the anomaly propagation paths. For example, Figure 2(a) shows four metrics ($m_1 - m_4$) collected from a microservices system, and Figure 2(b) shows a potential causal graph, where m_1 is the source of the anomaly.

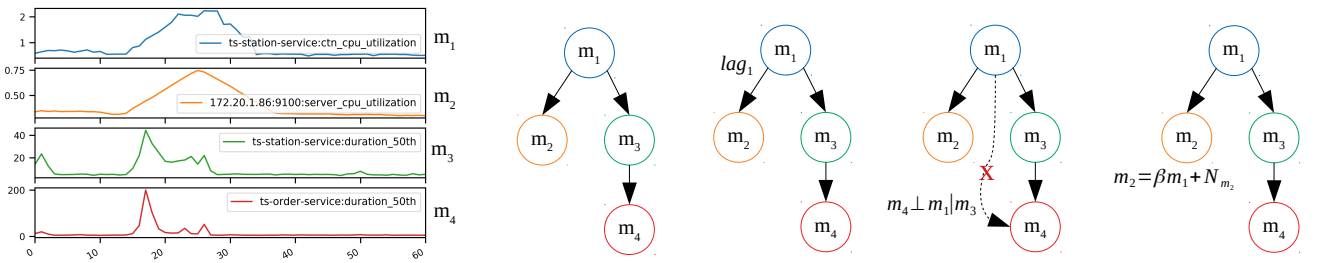
B. Overview of causal inference methods

There is an extensive study on inferring causal relations from observational data with CI methods in [5]. Here we give a brief introduction of causal inference techniques.

Granger causality: Granger causality (GC) is one of the most common methods for inferring the causal relationship between time-series based on temporal precedence. A time series $X = \{x_1, x_2, ..., x_t, ...\}$ 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 y_t of Y in comparison to predicting it based on the past values of Y alone. For example, metric m_1 is said to G-cause m_2 in Figure 2(c) if lag_1 is detected between them.

GC [6] was initially formalized as a prediction model using vector AutoRegressive (VAR). However, the used regression model assumes linearity and stationarity, and the accuracy of the causal inference is highly affected by predefined model order (which specifies how many previous time points are taken into account in the regression). There are some extensions of the basic GC concept, such as modeling nonlinear dependencies with transfer entropy, including multi-variable in GC, etc. Nevertheless, GC is limited to lagged causal dependencies and has known deficiencies in contemporaneous effects in sub-sampled time series [7].

Causal network learning algorithms: One of the most advanced theories with widespread recognition in discovering causality is the Causal Bayesian Network (CBN), which uses a Directed Acyclic Graph (DAG) to represent the causal relationships. There are two main approaches for learning a DAG: score-based approaches and constraint-based approaches. GES [8] is one score-based approach. It starts with an empty graph, then greedily searches a Markov equivalent class that maximizes a score function. The PC algorithm [9] is one of the state-of-the-art constraint-based approaches. It starts with a complete, undirected graph and recursively deletes edges based on conditional independence tests. For example, In Figure 2(d), the edge between metric m_1 and m_4 is deleted as $m_4 \perp m_1 | m_3$. Once the skeleton is obtained with



(a) Observational metrics (b) Causal graph (c) Granger causality (d) PC algorithm (e) Structural causal model

Figure 2: Overview of causal inference techniques.

conditional independence tests, edges are oriented with D-separation.

Learning a DAG from data is highly challenging and complex as the number of possible DAGs is super-exponential to the number of nodes. When applied to a high dimensional dataset, the runtime of the PC algorithm is exponential to the number of nodes in the worst case. But if the true underlying DAG is sparse, which is often a reasonable assumption, the runtime is reduced to polynomial run. There are some other methods, like FCI [9], which accounts for unobserved direct common causes and can still partially identify which links must be causal. PCMCI [10] incorporates time-order as a constraint (causes precede effects) and utilizes a set of causal orientation rules to identify causal directions.

Structural causal model (SCM): The key idea of SCM is to use the invariance of structural equations without committing to a specific functional form [11]. It assumes that the value of each variable is a deterministic function of its direct causes in the graph and some independent factors, like unmeasured disturbances. A general SCM is defined as Equation 1, where Y is the effect, X is the direct cause of Y, N_y is an independent factor, f_y is a function which links a variable to its direct causes can be any whatsoever.

$$Y = f_y(X, N_y). \tag{1}$$

Based on the concept of structural causal models, a set of variants are proposed to capture the underlying cause-effect relations. LiNGAM [12] assumes a linear function as: $Y = \beta X + N_y$, where X is independent non-Gaussian noise N_y. LiNGAM learns β such that X and $Y - \beta X$ are independent by applying independent component analysis (ICA). In our example Figure 2(e), the effect metric m_2 can be defined as a function of causal metric m_1 with an Non-Gaussian noise N_{m2}. ANM [13] extends the LiNGAM model and assumes that effect is a function of cause: $Y = f_y(X) + N_y$, where $X \perp\!\!\!\perp N_y$. The function is trained to map between m_2 and m_1, then it tests whether X and $Y - f_y(X)$ are independent using kernel independence tests.

III. Methodology

In this section, we introduce our experiment design which includes three research questions (Section III-A), CI-based performance diagnosis framework (Section III-B), selected CI methods (Section III-C), and fault injection experiments (Section III-D).

A. Experimental Design

To understand the performance of CI techniques on locating root causes in microservices, we first formulate three research questions, which demonstrate several dimensions of the challenges (i.e, numerous metrics and heterogeneous anomaly patterns) in microservice performance diagnosis; To cover the challenge of a wide range of root causes, we conduct a set of experiments by injecting three types of anomalies into multiple different services. Next, we evaluate the performance of CI methods according to the three research questions based on the metrics data collected from microservice benchmarks. Lastly, we study the computation overhead and scalability of selected CI methods. The details of our experiment design are shown in Table I. The three research questions are formulated as follows:

RQ1: *How do CI techniques perform on pinpointing the faulty service that initiates the performance issue* (coarse-grained diagnosis)?

To get localize the faulty service that initiates the performance issue, we apply CI methods on response times of all services to construct the anomaly propagation among services. The number of metrics is the same as the number of services, and the heterogeneity of anomaly patterns in response times is relatively low. In order to understand the performance of CI methods against different numbers of metrics in a low-level heterogeneity, we compare the performance of CI methods under three types of feature reduction, including no feature reduction, feature reduction, and ideal feature reduction. Moreover, we compare CI methods to a non-CI method which uses the domain knowledge of service dependency graph.

RQ2: *How do CI techniques perform on locating the culprit metric that contributes to the faulty service's abnormality with giving the faulty service* (fine-grained diagnosis with giving the coarse-grained cause)?

We assume the faulty service is known, which can be satisfied with our previous work [14] and other faulty service localization methods [3], [15], then apply CI methods on metrics exposed by the faulty service to identify the culprit metric that leads to the service's abnormality. The number of metrics is relatively low, but the heterogeneity among them increases as different types of metrics are likely to have different anomaly patterns.

Table I: Research questions, metrics, benchmarks, feature reduction, non-CI comparison, and results

Research question	Metrics	Benchmark	Feature reduction	Non-CI comparison	Results
RQ1	response times of all services	sock-shop	N	Y	Section IV-B
			Y		
			Ideal		
RQ2	metrics of the faulty service	sock-shop	N	N	Section IV-C
RQ3	metrics of all services	sock-shop	N	N	Section IV-D
			Y		
Overhead	metrics of RQ3	sock-shop	Y	N	Section IV-E1
Scalability	metrics of RQ1	train-ticket	Y	Y	Section IV-E2
	metrics of RQ3			N	

RQ3: *How do CI techniques perform on locating the culprit metric from all available metrics* (fine-grained diagnosis)?

We use all observable metrics to pinpoint the culprit metric that initiates the anomaly. The number of metrics is n times larger than the number of metrics in *RQ2*, where n is the number of services. Meanwhile, the heterogeneity of metrics is in a high level as diverse anomaly patterns exist not only in different types of metrics but also in heterogeneous services. With a high-level heterogeneity, we also compare the performance of CI methods against feature reduction.

B. Performance Diagnosis Framework

Figure 3 shows the framework for performance diagnosis with CI techniques. The input of the framework is metrics corresponding to the research questions (metrics for research questions are listed in Table I) with an optional feature reduction. Next, each CI method is employed to construct an anomaly propagation graph. Finally, it ranks the nodes in the graph and outputs a ranked list of potential root causes, where the top items have the highest probability to be the root causes. Based on the input metrics, the output root causes can be coarse-grained and fine-grained.

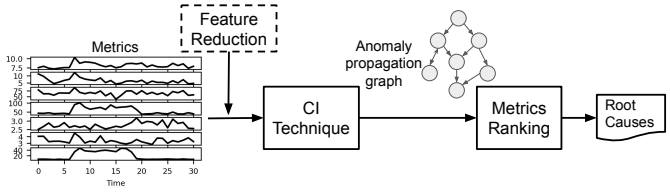


Figure 3: Performance diagnosis framework with CI techniques.

In our evaluation, we implement feature reduction with an unsupervised machine learning algorithm called Birch clustering [16] for identifying abnormal metrics, and utilize the PageRank graph centrality algorithm, which is commonly used in the state-of-the-art performance diagnosis approaches, to rank the root causes. Notably, we evaluate CI techniques with a basic ranking method as our focus is on comparing the CI techniques. For the actual application of such techniques, the performance can be improved with advanced ranking methods [14], [17].

C. Causal Inference Techniques

The objective of this work is to understand the performance of the class of CI techniques on microservice performance diagnosis. To achieve this goal, we select six representative CI methods from three categories, namely Granger causality, causal network learning algorithms, and structural equation models (SCM), based on their popularity, assumptions, and availability of implementations. Notably, we use the pure form of these methods rather than the tailed ones used in existing approaches [15], [17] as we aim to understand the performance of different types of CI methods on microservice performance diagnosis in a general way, rather than to achieve a high accuracy of diagnosis. The chosen CI methods are as follows:

- *Granger causality (GC)*: GC [6] is a popular CI method that has been adopted by many performance diagnosis systems [4], [18] to build the causal graph. We use χ^2 as the Granger causality test.
- *PC algorithm with partial correlation (PC-corr)*: PC-corr is a version based on PC algorithm [19] that uses Fisher's z-transformation of the partial correlation to test the (conditional) independence.
- *PC algorithm with kernel (PC-kernel)*: PC-kernel is a version of PC algorithm that uses kernel-based independence criteria [20] which can deal with non-linear causal-effect relationships and non-Gaussian noise. We use the Hilbert-Schmidt independence criterion to test independence.
- *Greedy Equivalence Search (GES)*: GES [8] is a score-based CI method. We use "obs" which is an ℓ_0-penalized Gaussian maximum likelihood estimator based on BIC score in the implementation.
- *Causal Additive models (CAM)*: CAM [21] is an SCM method which identifies causal relations by fitting an additive SEM with Gaussian error, where the causal-effect relationships can be non-linear.
- *Linear Non-Gaussian Acyclic Model (LiNGAM)*: LiNGAM [12] is an SCM method that assumes linear causal relationships but with non-Gaussian disturbance.

We conclude the chosen CI methods and their hyperparameters in Table II. In addition, we implement a non-CI method named *Corr* for comparisons, which uses the service dependency graph directly and incorporates the anomaly symptoms by setting weights with correlation coefficients.

Table II: The chosen causal inference methods and their hyperparameters

Method	Category	Assumptions	Hyperparameters	References
GC	Time-lag	Linearity	AIC criteria for lag selection; maxlag = 30s; χ^2 test for granger causality; p-value = 0.05.	package statsmodels [22]
PC-corr	Constraint-based	Gaussianity; Linearity	CItest = gaussian, alpha=0.01	Causal discovery toolbox [23]
PC-kernel	Constraint-based	Non-Gaussianity; Non-linearity	CItest = hsic_gamma, alpha=0.01	Causal discovery toolbox [23]
GES	Score-based	Gaussian; Linearity	score=obs	Causal discovery toolbox [23]
CAM	SCM	Gaussianity; Non-linearity	score=nonlinear, cutoff=0.001, sel-method=gamboost, pruning=False, prunmethod=gam	Causal discovery toolbox
LiNGAM	SCM	Non-Gaussianity; Linearity	nonparametric; max_iter=1000	ICALiNGAM in python package lingam [24]

D. Benchmarks, Injected Faults, and Evaluation Metrics

The evaluation is based on metrics data collected from fault injection experiments on microservice benchmarks.

Benchmarks: We select two representative microservice benchmarks named sock-shop[1] and train-ticket[2] (one of the largest microservice benchmarks), which simulate an e-commerce website for selling socks and a train ticket booking system, respectively. Sock-shop consists of thirteen microservices and train-ticket has forty-one microservices. Both of them are polyglot (e.g., Java, golang, Node.js, etc) and intercommunicate using REST over HTTP. Comparing to sock-shop, train-ticket has longer propagation paths and exposes a larger number of metrics.

Injected Faults: We inject three types of faults that are commonly used in the evaluation of the state-of-the-art performance diagnosis approaches [3], [4] to different microservices in the benchmarks: (1) Latency, we use a traffic control tool named *tc*[3] to delay the network packets; In our experiment, we inject 200 milliseconds delay to each microservice; (2) CPU hog, we use *stress-ng*[4], a tool to load and stress compute system, to exhaust CPU resources. For non-compute intensive microservices, like *payment* in sock-shop, we exhaust its CPU heavily with 99% usage and 95% for other microservices; (3) Memory leak: we use *stress-ng* to allocate memory continuously. For memory-intensive microservices, like *carts* and *orders* in sock-shop, a high memory usage introduces a high CPU usage, therefore we only provision 1VM to exhaust memory resource, and 2 VMs for other microservices.

In order to inject the performance issues to microservices, we update the existing Docker images by deploying the above faults injection tools and trigger the anomalies by running the corresponding commands inside the container. Each anomaly lasts 1 minute and the system has 5 minutes to cold down before another injection. To increase the generality, we repeat the injection process 3-5 times for each fault and service.

Evaluation Metrics: We quantify the performance of CI methods on diagnosing the root cause with two metrics: precision at top k (PR@k) and average precision at top k (AP@k). These two evaluation metrics are defined as follows:

- *Precision at top k* denotes the probability that the top k results given by an algorithm include the real root cause, denoted as $PR@k$. A higher $PR@k$ score, especially for small values of k, represents the algorithm correctly identifies the root cause. Let $R[i]$ be the rank of each cause and v_{rc} be the set of root causes. We now define $PR@k$ for 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 an algorithm with an average of PR@k:

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

IV. Experiment Results

In this section, we present the fault injection experiments setup, the results of the three research questions, and the discussion about the computation overhead and scalability of the selected CI techniques.

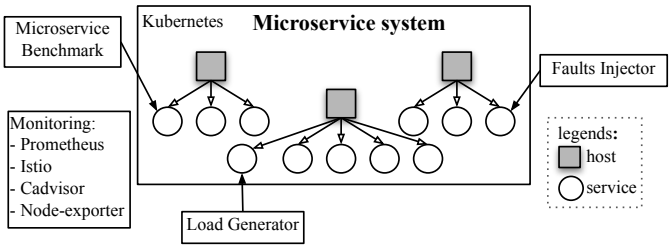
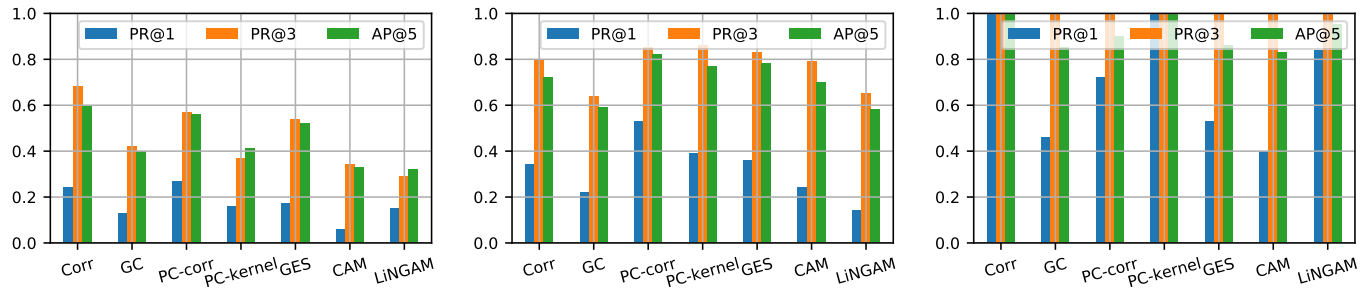


Figure 4: An overview of our experimental testbed.

A. Testbed

We create a testbed using Kubernetes where we deploy a microservice benchmark and a set of monitoring tools, as shown in Figure 4. In our Kubernetes cluster, there is one master node and multiple worker nodes (4 for sock-shop and 6 for train-ticket), including one worker node is dedicated for data collection and the rest for the microservices. Each worker node in the cluster consists of 4 vCPU and 15 GB memory. In addition, one virtual machine (6 vCPU and 12 GB memory) outside the cluster runs the load generator. In our deployment, we limit the CPU resource to 1 vCPU and memory to 1 GB

[1]Sock-shop - https://microservices-demo.github.io/
[2]Train-ticket - https://github.com/FudanSELab/train-ticket
[3]tc - https://linux.die.net/man/8/tc
[4]stress-ng - https://kernel.ubuntu.com/ cking/stress-ng/



(a) No feature reduction (b) Feature reduction (c) Ideal feature reduction

Figure 5: The results of PR@1, PR@3, and AP@5 for coarse-grained diagnosis against feature selections.

and set the replication factor to 1 for the microservice where faults are injected.

Load Generator: We develop a closed-loop load generator using Locust[5], a distributed, open-source load testing tool that simulates concurrent users in an application for each benchmark. We customize the workload to reflect the behavior of real users. For example, in sock-shop, more requests are sent to the entry points *front-end* and *catalogue*, and fewer to service *shopping*, *carts*, *user* and *orders*. In total, we provision 450-600 queries per second to both benchmarks.

Data Collection: We collect metrics from multiple layers of the testbed with a set of monitoring tools, including service mesh Istio[6] for microservices, Cadvisor[7] for containers, and node-exporter[8] for server nodes. Meanwhile, Prometheus[9] is deployed to collect metrics from above tools and is configured to scrape metrics every 5 seconds, which is the highest monitoring frequency feasible in our testbed with the existing versions of the used tools.

B. RQ1: How do CI techniques perform on coarse-grained diagnosis?

We evaluate the performance of CI techniques on locating the faulty service that initiates the performance issues based on the response times of all services. We first take all the metrics without any feature reduction as the input of CI techniques, and obtain the diagnosis result in terms of PR@1, PR@3, and AP@5 shown in Figure 5(a). Overall, all the methods cannot identify the faulty services well (with a maximum of 27% in PR@1). This is because services are inter-dependent not only in abnormal status but also normal status. These normal dependencies in metrics introduce extra causal links in the graph, which renders the graph centrality algorithm unable to get the precise root cause.

Next, we apply a feature reduction which results in an F1-score of 0.6 in anomaly detection. The diagnosis result shown in Figure 5(b) shows that CI technique with feature reduction improves on average 77.4% in PR@3, outperforming this scenario with no feature reduction by 31.6%. In addition, we can see that constraint-based methods achieve a higher performance than non-CI method *Corr*, like *PC-corr* achieves an improvement of 19% in terms of PR@1 over *Corr*. This is because feature reduction with an F1-score of 0.6 includes false positives, resulting spurious propagation paths in *Corr*. In contrast, CI methods can eliminate parts of the spurious paths, thus increasing the accuracy.

Finally, we evaluate the performance of CI methods with an ideal feature reduction, where only the expected anomalies, including the faulty service and its upstream services, are used to construct the anomaly propagation graph. The performance of all methods is shown in Figure 5(c). Overall, all the methods achieve a higher performance when only true-positives are detected as anomalies, on average, achieving 70.7% in PR@1, improving 39% over feature selection with an F1-score of 0.6. Particularly, both *Corr* and *PC-kernel* achieves 100% precision.

We note that the results reported here can be improved if advance methods are applied according to the state-of-the-art methods, like using metrics from multiple layers to score the abnormality of services and assigning attributes to edges and nodes before ranking that have been done in our previous work MicroRCA [14].

Summary: Neither CI methods nor the non-CI method based on service dependency graphs can identify the faulty services well without feature reduction, as causal relations exist among both normal and abnormal metrics. However, their performance can be improved if feature selection can be applied before causal inference. In particular, *PC-kernel* and the non-CI method *Corr* achieve 100% in precision when only true-positives are detected as anomalies. More importantly, constraint-based CI methods, like *PC-corr*, are more robust than knowledge-based non-CI method to false positives.

C. RQ2: How do CI techniques perform on fine-grained diagnosis with giving the faulty service?

To address this research question, we apply CI methods to the faulty service's relevant metrics and evaluate their performance on locating the culprit metrics that contributes the service abnormality. Table III shows the performance of each CI method on locating the culprit metric on different

[5]Locust - https://locust.io/
[6]Istio - https://istio.io/
[7]Cadvior - https://github.com/google/cadvisor
[8]Node-exporter - https://github.com/prometheus/node$_exporter$
[9]Prometheus - https://prometheus.io/

Table III: The performance of CI methods on fine-grained diagnosis with giving faulty service against different type of anomaly.

Methods	GC	PC-corr	PC-kernel	GES	CAM	LiNGAM	average
Latency							
PR@1	**0.24**	0.15	0.15	0.21	0.09	0.21	0.175
PR@3	**0.56**	0.41	0.44	0.41	0.38	0.41	0.435
AP@5	**0.55**	0.45	0.49	0.44	0.36	0.44	0.455
CPU Hog							
PR@1	0.23	**0.63**	**0.73**	0.17	0.2	0.17	0.355
PR@3	0.70	**0.80**	**0.83**	0.73	0.5	0.73	0.715
AP@5	0.66	**0.85**	**0.87**	0.63	0.45	0.63	0.68
Memory Leak							
PR@1	**0.32**	0.11	0.11	0.18	0.21	0.18	0.19
PR@3	0.57	0.57	0.50	**0.61**	0.39	**0.61**	0.54
AP@5	**0.56**	0.51	0.49	**0.54**	0.46	**0.54**	0.517

types of anomalies in terms of PR@1, PR@3, and AP@5 when no feature reduction is applied. We can see that Granger causality (GC) performs better than other methods on latency issues. This is because latency anomalies propagate to other resources through service performance degradation, which can be reflected as time-lags among metrics, which satisfies the assumption of GC.

Regarding CPU hog issues, most of the CI methods identify the root cause with a higher performance except the CAM method. This is because CPU issues reflect themselves as strong correlations between cause and effect metrics and weak correlations with non-causal metrics. Figure 6 shows the maximum and minimum coefficient of determination r^2 of linear regression (r_max, r_min) and Pearson correlation coefficients (p_max, p_min) between the culprit metric and other metrics. We can see that there is a significant difference between the maximum and minimum values (95.7% of p_max and 12% of p_min). Due to these obvious differences between causal and non-causal metrics, it is easier to diagnose the root cause. Meanwhile, due to this linearity between cause and effect metrics in CPU hog issues, the CAM method performs poorly for this type of anomalies as it models nonlinear causal relations. Conversely, we can see that latency and memory leak issues yield smaller differences between causal and non-causal metrics, resulting in lower performance for those methods that assume linearity.

For memory leak issues, we can see that LiNGAM and GES perform better than other methods in terms of PR@3.

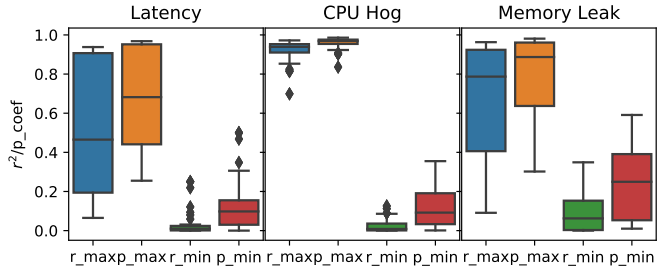


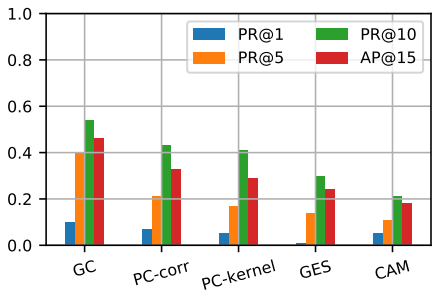
Figure 6: R-squared and Pearson correlation coefficient between culprit metric and other metrics of different types of anomaly.

This is because memory leak issues manifest themselves in multiple resource metrics (e.g., memory utilization and CPU utilization), which introduces contemporaneous effects that can be difficult to identify for other CI methods. LiNGAM is a model that can capture such effects, thus it performs better for memory leak issues.

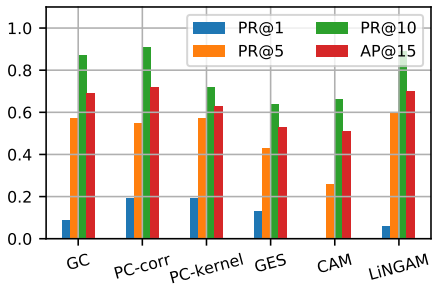
Summary: Anomaly symptoms in service relevant metrics vary with the type of anomaly, which violate or satisfy the assumptions of a specific CI method. To exemplify, Granger causality performs well on latency issues, SCM method LiNGAM performs well on instantaneous effects in memory leak issues, and most of the CI methods, particularly the PC-algorithm, perform well for correlated anomaly symptoms caused by CPU hogging. On average, PC-algorithm has a better performance on the three types of anomalies.

D. RQ3: How do CI techniques perform on fine-grained diagnosis with all observational metrics?

We address this research question by evaluating the performance of CI on identifying the culprit metrics from all observed metrics. Figure 7(a) shows the performance of culprit metric diagnosis of each CI method in terms of PR@1, PR@5, PR@10, and AP@15, where no feature reduction is applied. In this experiment, we use a larger $k = \{1, 5, 10, 15\}$ in the evaluation metrics comparing to RQ1 and RQ2 ($k = \{1, 3, 5\}$). This is because the number of potential root causes is much larger, and a small k cannot differentiate the performance of CI methods well. We note that LiNGAM is not included when the number of metrics exceeds the sample size, for which the method is not defined.



(a) No feature reduction

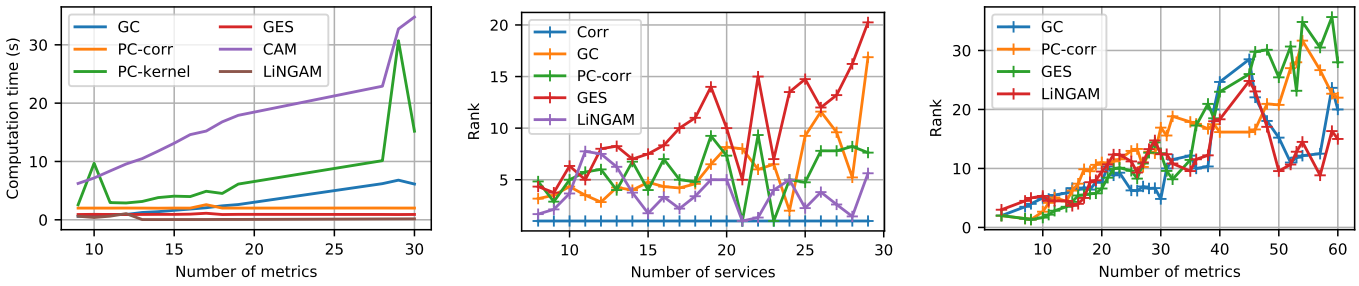


(b) Feature reduction

Figure 7: The results of PR@1, PR@5, PR@10, and AP@15 for fine-grained diagnosis against feature reductions.

Table IV: The performance of CI methods on different ranges of metrics in terms of PR@3.

RQ	Metrics	GC	PC-corr	PC-kernel	GES	CAM	LiNGAM	Average
RQ1	response times of all services	0.47	**0.62**	0.47	0.25	0.41	0.25	0.45
RQ2	metrics of culprit service	**0.61**	0.59	**0.59**	**0.58**	**0.42**	**0.58**	**0.56**
RQ3	metrics of all services with filtering	0.45	0.4	0.43	0.23	0.15	0.26	0.32



(a) Computation overhead (b) Coarse-grained diagnosis (c) Fine-grained diagnosis

Figure 8: Computation overhead, the rank of root causes against the number of metrics or services.

We can see that most of the methods cannot locate the culprit metric well, with an average of 5.6% in PR@1, and 20.6% in PR@5. This is because (1) higher-dimensional features exaggerate the difficulty to infer a precise anomaly propagation graph; (2) anomalies propagate across services and also their metrics, resulting in many instantaneous effects that are difficult to discover; (3) due to the heterogeneity of services, their anomalous metrics commonly manifest diverse symptoms. It is hard to capture various causal relations with one CI method; (4) normal patterns of metrics also introduce unexpected inference.

Next, we evaluate the performance of CI methods with the feature reduction. Figure 7(b) shows the performance of CI methods on reduced metrics. Overall, all the CI methods achieve an improvement after the feature reduction with achieving an average 5.4% in PR@1 and 29.1% in PR@5 more than no feature selection. Particularly, GC, PC-corr, PC-kernel, and LiNGAM achieve an average of 57% in terms of PR@5. With the feature reduction, some unexpected and uncorrelated patterns can be removed from metrics, thus reducing the inference to causal graph learning. Additionaly, we notice that LiNGAM achieves the highest rank in PR@5 with 60%, which indicates that LiNGAM has a higher performance in identifying fine-grained causes in microservice systems.

Lastly, we compare the performance of CI methods on different groups of metrics corresponding to our three research questions. Table IV shows the performance in terms of PR@3 of each CI method. We can see that the performance of CI methods on dealing with metrics of all services (RQ3) is lower than others, as metrics in RQ3 include diverse anomaly patterns not only from service-relevant metrics but also from heterogeneous services. In comparison, most CI methods achieve a higher performance in identifying the culprit metric from metrics exposed by faulty service.

Summary: It is difficult to identify fine-grained causes from all metrics, which is a mixture of many normal and abnormal metrics (20.6% in PR@5). The performance of CI methods can be improved with the feature reduction, but it does not rank the root cause in the top 5 of the list well (49.7% in PR@5). Instead, a drill-down approach that identifies the faulty service first, then pinpoints the culprit metric that attributes to the faulty service, is more promising to obtain the fine-grained causes.

E. Computation overhead and scalability

Lastly, we discuss the overhead of the selected CI methods and their scalability to the number of microservices and metrics, which are important to an online and scalable microservice performance diagnosis.

Overhead: We analyze the computation overhead of each CI method, varying with the number of metrics it handles. Figure 8(a) shows the computation time of the six evaluated CI methods (running on the same Intel Core i7 version server with 8-core CPU and 16 GB memory) against the number of metrics (from 10 to 30, and with 5 minutes of data gathered from each metric, in total of 60 data points). We can see that the computation time of CAM and PC-kernel increase dramatically with the number of features, and GC also becomes slightly slower when the number of metrics increases.

Scalability: We evaluate the scalability of CI methods on the one the largest benchmark, named train-ticket, which has longer propagation paths and a larger number of metrics.

We conduct the same fault injection experiments on the service *station* in benchmark train-ticket, which has the longest propagation paths. Then we apply four computation efficient CI methods, including GC, PC-corr, GES, and LiNGAM on metrics corresponding to RQ1 and RQ3.

We evaluate the performance of each CI method by measuring the rank of root cause against different number of metrics, which are produced by changing the threshold of the feature reduction. Figure 8(b) and (c) show the average rank of root cause (lower is better) against a different number of service-level metrics and metrics of all services. We can

see from Figure 8(b) that the performance of *Corr* with the service dependency graph by far outperforms all CI methods. This is because the service *station* has a high in-degree (upstream services), which results in high ranking accuracy in *Corr*'s graph centrality algorithm. Meanwhile, we can see that LiNGAM identifies the culprit service in the top 5 often as the number of metrics increases. However, from Figure 8(c) we can see that all CI methods have increasing ranks of root causes with the increasing number of metrics, which is due to the same issue of diverse anomaly symptoms mentioned in Section IV-D.

Summary: The computation overhead of some CI methods, such as PC-kernel and GES, increases drastically with the number of metrics. When metrics have similar patterns, like service-level metrics, LiNGAM can identify the faulty service well even when the number of microservices is high. However, identifying the root causes from all metrics in the system is difficult due to the high-level heterogeneity.

V. Related Work

In recent years, many approaches have been proposed to diagnose problems in computer network [25], clouds [26] and microservices. Overall, these approaches employ machine learning [27], [28], pattern recognition [29], graph-based [15] methods with observability from the system, such as logs [30], requests execution tracing data [31], and metrics [4], to diagnose the root cause. Here we review the graph-based approaches based on CI techniques as follows.

Graph-based approach can not only identify the root cause but also provide visibility of the issue with a visualized graph. This kind of approach identifies the root cause by constructing a causal graph from observational data, then inferring and ranking the causes based on the graph. Microscope [3] and MicroRCA [14] pinpoint a coarse-grained root cause by modeling the anomaly propagation through service dependency graph and service co-location.Both leverage the ground truth of the service invocation paths and service deployment by parsing network information or metrics from multi-layer in MSA.

In addition, graph-based approaches are widely employed to locate root cause at a fine granularity [4], [15], [17], [18], [32] by using causal inference methods to construct the causal graph. Sieve [18] and Loud [4] construct causal graphs among key performance indicates (KPIs) with Granger causality tests. However, Granger causality tests assume time-lag and linearity or nonlinearity are embedded in the cause-effect metrics, which is quite restrictive for observational metrics in microservice systems. Causeinfer [15] and [32] discover the causal graph with PC algorithm based on conditional independence test. Even though Causeinfer uses a cross-entropy G-squared independence test to overcome the limitations of strict assumptions that other independence tests require, contemporary effects are difficult to identify with CI tests. Moreover, the causal-effect relations are not sparse among metrics which would add more overhead to the PC algorithm. A variant of the PC algorithm which considers the time-order of metrics is used in MicroCause [17] to identify a causality graph of metrics. However, it would inherit both the limitations of time-lag based method, and the high computation burden of the PC algorithm. To illustrate these issues and compare methods, we in this paper provide a benchmark for different types of CI techniques on identifying the root causes of different types of anomalies. We note that our experiment results can give some insights for improving the efficiency of CI methods on performance diagnosis.

VI. Conclusions

In this paper, we evaluate the performance of CI techniques for microservice performance diagnosis using six representative CI methods from three categories. Our evaluation based on a set of fault injection experiments shows that 1) CI methods need to incorporate anomaly symptoms through anomaly detection or anomaly score to achieve high performance. In addition, constraint-based CI methods are more robust to false positives in anomaly detection than the domain knowledge-based non-CI method; 2) The heterogeneity in service relevant metrics depends on the type of anomaly. This makes the performance of CI techniques vary with the types of anomalies, among which PC-algorithm performs well for correlated anomaly symptoms. 3) To identify fine-grained root causes that indicates both the faulty service and its abnormality, a drill-down approach that identifies the faulty service first then looks into the detailed information from its relevant metrics, is better than inferring from all metrics. 4) With an increasing number of services or metrics, the runtime of CI method GES and PC-kernel tends to increase drastically, and the precision of fine-grained diagnosis decreases. However, CI method LiNGAM can still perform well in identifying the faulty service that initiates the performance issue.

Based on our findings, we propose the following research directions to improve the efficiency of the application of CI techniques for microservice performance diagnosis:

1) Automatic search or online tuning of CI methods: It is hard to cover all types of anomalies with one specific CI method due to the heterogeneous anomaly pattens. Therefore, an automatic selection or online tuning of CI methods based on the characteristics of the observational data can help improve the performance of locating root causes from a wide range.

2) Learning causal graph with soft intervention: The constructed causal graph can be improved with the data collected from soft intervening on microservice systems, such as restart or scale-out. However, before applying any interventions, a risk estimation of the intervention is required.

3) Employing expert knowledge as much as possible: Collaborating CI methods with expert knowledge or ground truth, like service dependency, can eliminate inference from irrelevant or duplicated features, thus increasing the accuracy of causal graph learning and performance diagnosis.

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] 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] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[3] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *Service-Oriented Computing*, C. Pahl, M. Vukovic, J. Yin, and Q. Yu, Eds., 2018, pp. 3–20.

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

[5] A. Nichols, "Causal inference with observational data," *The Stata Journal*, vol. 7, no. 4, pp. 507–541, 2007.

[6] C. W. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica: journal of the Econometric Society*, pp. 424–438, 1969.

[7] P. Spirtes and K. Zhang, "Causal discovery and inference: Concepts and recent methodological advances," in *Applied informatics*, Springer, vol. 3, 2016, p. 3.

[8] D. M. Chickering, "Optimal structure identification with greedy search," *Journal of machine learning research*, vol. 3, no. Nov, pp. 507–554, 2002.

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

[10] J. Runge, P. Nowack, M. Kretschmer, S. Flaxman, and D. Sejdinovic, "Detecting and quantifying causal associations in large nonlinear time series datasets," *Science Advances*, vol. 5, no. 11, 2019.

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

[12] S. Shimizu, P. O. Hoyer, A. Hyvärinen, and A. Kerminen, "A linear non-gaussian acyclic model for causal discovery," *Journal of Machine Learning Research*, vol. 7, no. Oct, pp. 2003–2030, 2006.

[13] P. Hoyer, D. Janzing, J. M. Mooij, J. Peters, and B. Schölkopf, "Nonlinear causal discovery with additive noise models," *Advances in neural information processing systems*, vol. 21, pp. 689–696, 2008.

[14] 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.

[15] 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*, 2014, pp. 1887–1895.

[16] 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.

[17] 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)*, 2020, pp. 1–10.

[18] 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.

[19] M. Kalisch and P. Bühlmann, "Estimating high-dimensional directed acyclic graphs with the pc-algorithm," *Journal of Machine Learning Research*, vol. 8, no. Mar, pp. 613–636, 2007.

[20] A. Gretton, R. Herbrich, A. Smola, O. Bousquet, and B. Schölkopf, "Kernel methods for measuring independence," *Journal of Machine Learning Research*, vol. 6, no. Dec, pp. 2075–2129, 2005.

[21] P. Bühlmann, J. Peters, J. Ernest, *et al.*, "Cam: Causal additive models, high-dimensional order search and penalized regression," *The Annals of Statistics*, vol. 42, no. 6, pp. 2526–2556, 2014.

[22] *Granger causality tests*. [Online]. Available: https://www.statsmodels.org/stable/generated/statsmodels.tsa.stattools.grangercausalitytests.html, (accessed: 27.12.2020).

[23] D. Kalainathan, O. Goudet, and R. Dutta, "Causal discovery toolbox: Uncovering causal relationships in python.," *Journal of Machine Learning Research*, vol. 21, no. 37, pp. 1–5, 2020.

[24] *LiNGAM*. [Online]. Available: https://github.com/cdt15/lingam, (accessed: 27.12.2020).

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] P. Zhou, Y. Wang, Z. Li, X. Wang, G. Tyson, and G. Xie, "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.

[31] 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*, New York, NY, USA, 2019, pp. 683–694.

[32] J. Qiu, Q. Du, K. Yin, S.-L. Zhang, and C. Qian, "A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications," *Applied Sciences*, vol. 10, no. 6, p. 2166, 2020.