Diagnosing Performance Issues in Microservices with Heterogeneous Data Source

Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li
School of Software and Microelectronics, Peking University
Beijing, China
{cjhou, jia.tong, yifanwu, li.ying}@pku.edu.cn

Jing Han
ZTE Corporation
Shanghai, China
han.jing28@zte.com.cn

***Abstract*—Microservices architecture is vulnerable to performance issues due to its highly fine-grained decomposition of an application. To diagnose performance issues in microservices, existing works utilize system metrics as the specific indicator and do a lot of heavy computation such as building service dependency graphs during the diagnosing process.**

To improve the effectiveness and efficiency of issue diagnosing, we propose PDiagnose, a practical approach using multiple data sources including metrics, logs and traces jointly to diagnose performance issues in microservices systems. Through combining lightweight unsupervised anomaly detection algorithms and vote-based issue localization strategy, PDiagnose is application-agnostic and can localize root cause indicators accurately. Our evaluation on two public-available datasets shows that PDiagnose can achieve an overall recall of 84.8%, outperforming the best baseline approach. Meanwhile, the diagnosis duration of PDiagnose is also promising.

***Index Terms*—anomaly detection, fault localization, KPI, microservice**

I. Introduction

Microservices architecture is a promising and popular paradigm of today's systems ([1]–[3]). With microservices architecture, a complicated application can be decomposed into fine-grained, lightweight, elastic and independently-maintainable microservices. Each light-weight microservice is responsible for one simple functionality, thus is easy to scale out. Therefore, system performance, scalability and flexibility are highly improved.

Meanwhile, the fine-grained decomposition of microservices makes the system vulnerable to performance issues. Today's microservice systems consist of tens of thousands of microservices. For example, Alibaba's cluster has been hosting more than 10,000 microservices since 2019 [4]. Each microservice can suffer from performance issues due to many unpredictable problems such as network failure, host issues, etc. In addition, the control flow and data flow to handle a single request can cross hundreds of microservices, the performance issue of every microservice will affect the whole request. Therefore, to ensure system performance and reliability, it is necessary to monitor and efficiently diagnose the performance issues of each microservice.

However, diagnosing the performance issues of microservice systems is not easy. First, thousands of microservices produce huge amount of monitoring data including KPIs (i.e. key performance indicators or metrics), system logs and request traces. Analyzing these data and finding abnormal behaviors from these data is time-consuming. Second, the heterogeneity of different microservices leads to different symptoms, which makes it hard to find generic anomaly patterns. Different technology stacks can be an advantage for microservices architecture, while those differences can also bring great challenges to the issue diagnosis problem. Third, complicated and varying dependencies among microservices are disastrous when a critical issue occurs. Typical large-scale microservices systems can update hundreds to thousands times a day and the microservice dependency relationships are highly dynamic. Therefore, diagnosing performance issues of microservices can be really cumbersome work. Fig. 1 shows an example of a request flow in a microservice system. A simple front-end request may cause an issue in a low-tier microservice, and further affect numerous upper microservices.

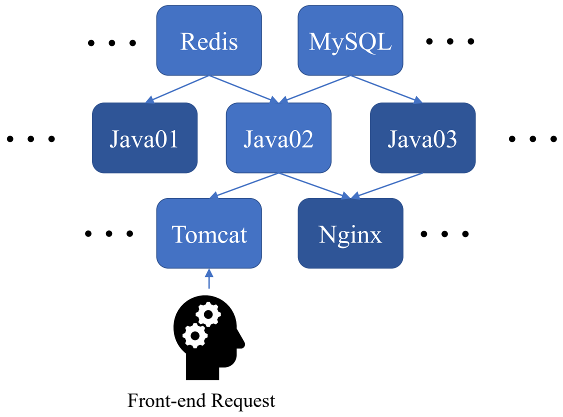

Fig. 1. A simple request depends on numbers of microservices. The affected microservices' failure may cross propagate among the system, making it more difficult to diagnose microservices systems.

Researchers have conducted many studies to improve the effectiveness and efficiency of issue diagnosing. Some recent works ([5]–[10]) used anomalous KPIs to construct a KPI dependency graph, which can represent the correlation between different KPIs and expose root cause indicators. Some other studies ([11], [12]) have taken inter-microservice topology into consideration and build dependency graphs with priori knowledge. Although these methods have achieved good performance in localizing microservices with issues, there are still some key limitations:

- **Building dependency graph is time-consuming.** Most of the existing dependency graph-based methods use the PC algorithm to construct dependency graph on KPIs. Unfortunately, the PC algorithm has exponential worst-case complexity. Thus, it is infeasible to apply dependency graph-based methods to large-scale microservice systems.
- **Detecting anomalies and localizing root causes independently bring cascade error.** Existing root cause localization methods rely on anomaly detection results to build dependency graphs and infer root causes. If anomaly detection approaches fail to discover and report issues, the performance of root cause localization will be significantly affected.
- **Using single source indicator degrades the diagnosis performance.** Modern large-scale systems collect tons of heterogeneous indicators every day, including KPIs, logs and traces. Nevertheless, existing studies focus on using single-source indicators to diagnose issues. For example, MicroRCA [11] and CloudRanger [7] only take KPIs into account. Missing crucial information from other indicators can potentially degrade the performance of issue diagnosis.

To address these limitations, this paper proposes PDiagnose (shortly for **P**erformance **D**iagnose), an application-agnostic approach utilizing KPIs, system logs and traces to diagnose microservices performance issues in real-time. We apply lightweight non-supervised anomaly detection algorithm on microservices KPIs. For logs and traces, we use general-purpose methods to capture suspicious microservices and corresponding logs. These data are collected and sent to an issue identification and localization module where false alarms are dropped and true root causes are localized. To identify if an issue is a real fault, we propose a novel identification approach considering the distribution of KPI anomalies. And finally, suspicious microservices and log entries obtained from traces and logs are utilized to localize the root cause microservices, KPIs and log entries.

To summarize, our contributions are as follows:

- We propose an effective microservices performance issue diagnosis approach, PDiagnose. While monitoring real-time KPIs, logs and traces, PDiagnose is able to localize the root causes to the extent (e.g. microservice ID, KPI names and raw log entries) that SREs (i.e. site reliability engineers) can easily explain and recovery operations can be instantly done.
- Our experiments on two public-available real datasets show that PDiagnose outperforms the baselines and the diagnosis efficiency is also promising.

The rest paper is organized as follows:Section II presents related work while section III elaborates the preliminaries of the problem and our approach. Section IV details our approach. Section V describes the experiments and results. We discuss our approach's limitation in Section VI and conclude the paper in Section VII.

II. Related Work

In the literature, KPI-based issue diagnosis has been well studied. Most of existing approaches ([5]–[12]) attempted to localize the root cause KPIs via construction the dependency graph automatically.

These graph-based approaches can be summarized in two main steps, graph construction and root cause localization on the graph. Among these approaches, MonitorRank [5], CauseInfer [6], CloudRanger [7] and Microscope [8] used PC algorithm [13] on anomalous KPIs to infer the dependency graphs, MicroRCA [11] utilizes microservice topology to build the graph, while FluxInfer [9] designed Weighted Undirected Dependency Graph (WUDG) algorithm to get the dependency graph. For the root localization step, different algorithms are applied to find the root cause on the graph. CauseInfer used depth first search (DFS) algorithm. MonitorRank and CloudRanger applied random walk algorithm. FluxInfer and MicroRCA proposed different improved versions of PageRank algorithm [14]. Although these approaches achieved good performance, the overhead of building the dependency graph cannot be ignored. Also, these approaches take off-the-shelf anomaly detection results as inputs and the detection error may propagate through different steps.

Researchers also proposed some other diagnosis approaches. Supposing root cause KPIs change earlier, PAL [15] ranked KPIs with their anomalous change start time to localize the root causes. Seer [16] used a deep learning model to identify anomalous microservice, then predict QoS violations and schedules microservices accordingly.

Extracting information from logs and traces to diagnose issues has also been a hot topic [17]. Graph-based models (e.g. LogSed [18]) first extract log templates from raw logs, then generate graphs and compare them with online logs to find issues. Some other work [19] quantified the frequency of different log templates and leveraged machine learning models to find anomalous logs. TraceAnomaly [20] learned normal trace patterns by offline training, and used online traces' anomaly score to identify if an issue happens. Our approach aims to effectively combine the abundant information from all of these heterogeneous data sources for better performance.

III. Preliminaries

As a basis of further discussion, it is necessary to clarify the specific meanings of KPIs, traces and logs before we describe our approach. KPIs, also known as metrics, are performance-related time series taken in successive order with equal space. Some typical examples of KPI include system CPU util, memory util, network throughput, etc. In contrary, traces are collected when a request is received and cross-microservice calls are required. A simple trace message quadruple $\langle Req, Caller, Callee, Duration\rangle$ consists of request ID, caller microservice ID, callee microservice ID, and the time span of the function call. By combining

TABLE I
NOTATIONS

Notation	Definition
s_i	Microservice i
$\{x\}, \{y\}, \{z\}$	Different KPI time series
x_t	Value of KPI x at time t
$a, \hat{a}$	Actual value a and predicted value $\hat{a}$
a, a'	Original value a and normalized value a'
$\langle Req, Caller, Callee, Duration \rangle$	A trace message including request ID, caller microservice, callee microservice and execution time
$L_{i,t}, SuspL_{i,t}$	A log entry from microservice i at time t, and suspicious log entry
$SuspS$	Suspicious microservice obtained from trace analysis
$Q = \{q_1, q_2, \ldots\}$	Anomaly KPI queue Q, containing anomaly KPI quadruples q_1, q_2, ...
$q = \langle t, s_i, x, A \rangle$	Anomaly KPI quadruple q, meaning KPI x of microservice i is anomalous at time t with anomaly score A
$q.t, q.s, q.x, q.A$	The timestamp, microservice, KPI and anomaly score of a anomaly KPI quadruple q

trace messages from the same request, an invocation chain can be obtained to better understand the behavior of the microservices systems. Logs show events recorded by applications and operating systems. As an important source of issue diagnosis, logs are usually human-friendly and informative. A log entry often contains event timestamp, event level (e.g. INFO, WARNING or ERROR) and event content. Despite their representation, KPIs, traces and logs all play important roles in issue diagnosis. We summarize the notations of these indicators used in our study in Table I.

IV. Approach

Fig. 2 gives an overview of our approach. From microservices systems, PDiagnose collects real-time data, including KPIs, system logs and traces. These indicators are processed in different components. Leveraging a lightweight anomaly detection algorithm with unsupervised learning on KPI time series, PDiagnose constructs an anomaly KPI queue containing anomalous KPI points. Meanwhile, traces and logs are analyzed to screen out irrelevant microservices. Continually, the most suspicious microservices and logs are updated. To identify a true occurred issue, the status of the anomaly KPI queue is carefully monitored. Once the pattern of anomaly KPI queue meets specific conditions, a true issue is believed to have occurred and the root cause diagnosis process is triggered. The root cause microservices is localized with features extracted from the anomaly KPI queue and the suspicious microservices sent by the trace analysis component. Finally, suspicious logs along with the anomaly KPI queue are utilized to localize the root cause indicators.

In the next subsections, we introduce the main components of PDiagonse: KPI anomaly detection, log/trace analysis and issue identification/localization in detail.

A. KPI Anomaly Detection

To obtain the anomaly KPI queue, we need to know that at each moment, whether a particular KPI becomes anomalous or keeps normal. To formulate the problem, we denote the KPI as a time series by $\{x_t\}$, where subscript t stands for the KPI collection time. Picking time window size T, our goal is to quantify the abnormality of latest data point x_t based on normal historical data series $\{x_{t-T}, x_{t-T+1}, ...x_{t-1}\}$.

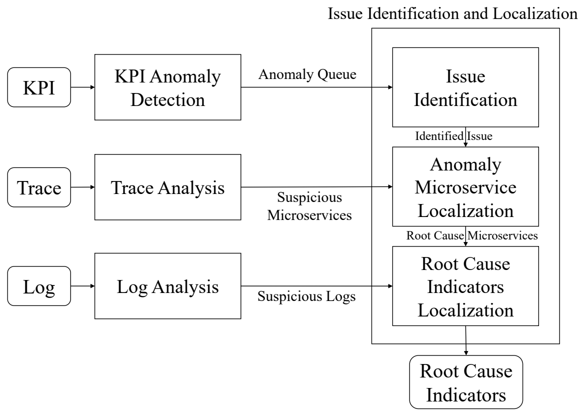

Fig. 2. PDiagnose Overview

Time series anomaly detection is a classical research field. Many effective methods have been proposed in this area and have achieved promising results ([21], [22]). However, in our case, the anomaly detection algorithm has to be efficient enough to handle thousands of KPIs at the same time and meanwhile be of great versatility in order to be adapted to different kinds of microservices systems.

We design an integrated anomaly detection algorithm to construct the anomaly KPI queue for further use. To capture anomalous fluctuations, Kernel Density Estimation (KDE) [23] and Weighted Moving Average (WMA) [24] are adopted as basic methods. As a non-parametric test approach, Kernel Density Estimation is able to estimate a random variable's probability density function $f_X(x)$ based on observed examples unsupervisedly. G refers to a kernel function, and in our case the Gaussian 1-D kernel is used in convenience. With T historical samples, a probability distribution is estimated:

$$f_X(x) = \frac{1}{T-1} \sum_{i=t-T}^{t-1} G(x; x_i) \quad (1)$$

Hence, the recent abnormality of a particular KPI $\{x_t\}$ can be measured with p_value from hypothesis test. We also use k-sigma rule to remove normal data points. Weighted Moving Average is another simple yet effective time series

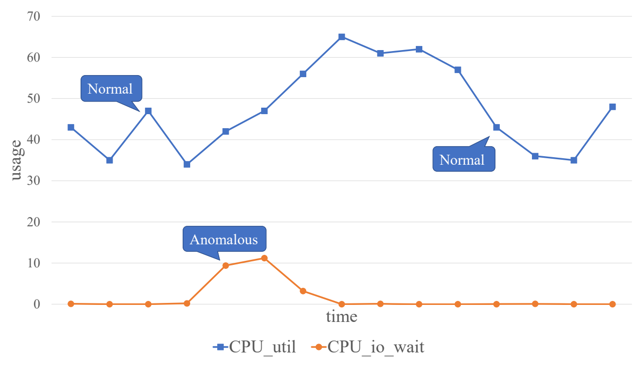

Fig. 3. Two CPU-related KPIs with different characteristics

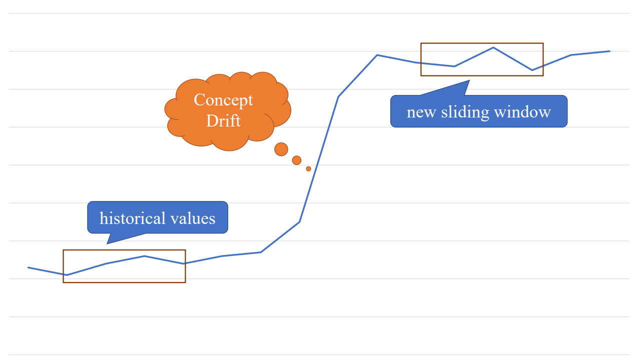

Fig. 4. The dynamic update of sliding window

forecasting approach. It utilizes weighted average of sliding window values to predict expected value $\hat{x}_t$. We took the difference between the real value x_t and expected value $\hat{x}_t$ as a quantitative indicator of a KPI's abnormality.

These methods work well for numeric KPIs. However, for some stable percentage indicators, even a small change might suggest an issue. For example, Fig. 3 shows that a server's CPU_util could possibly fluctuate between 30% to 70% due to daily peaks, while a slight change of CPU_io_wait from 0.1% to 10% could indicate a severe IO issue. Therefore, we apply mutation detection on percentage indicators. Specifically, we consider KPIs' first-order difference as a derived series, compare its relative range with the original time series, and the relative range's deviation Δ_R is regarded as an abnormality indicator of a series of percentages.

The overall anomaly score A of a KPI is an ensemble of the indicators evaluated above. It is defined as:

$$A = w_1 p\_value + w_2 |x_t - \hat{x}_t|' + w_3 \Delta_R' \tag{2}$$

We assign different weights to p_value, $|x_t - \hat{x}_t|'$, and the relative range's deviation, then normalize them to obtain the overall anomaly score, ranged from 0 to 1.

To calculate the abnormality indicators and the overall anomaly score, it is crucial to maintain the sliding window of size T. To ensure the historical values are irrelevant to anomalies, after an anomalous data point is found, the sliding window would just ignore this point until the next normal point is detected. When it occurs that K or more consecutive points from the same microservice and KPI are found to be anomalous, we consider there has been a concept drift [25] (e.g. a system update). Under this circumstance, we discard the current sliding window and take the latest T points as a new sliding window, shown in Fig. 4. Note that T and K are parameters and can be configured by SREs.

Finally, all the KPI data points with an anomaly score higher than a certain threshold will be collected to build the anomaly KPI queue Q. The weights and the anomaly score threshold can either be manually set or be learned from historical data with machine learning approaches like XGBoost [26]. A typical anomaly quadruple $q = \langle t, s_i, x, A \rangle$ in Q consists of anomaly timestamp t, microservice ID s_i, anomaly KPI name x and anomaly score A. The anomaly KPI queue will be useful to identify issues and localize root causes.

B. *Log and Trace Analysis*

Each microservice outputs a sequence of log entries. Each log entry is formulated as $L_{i,t}$, where i denotes a certain microservice, t denotes the print time of the log entry. When the microservice suffers from performance issues, some anomaly messages may be printed out in the log entries. Therefore, we match a few keywords that are such as "error", "problem", etc., in the log entries. As long as a log entry $L_{i,t}$ contains a keyword, we report s_i as suspicious microservice and $L_{i,t}$ as suspicious log entry to issue identification and localization component.

System tracing aims to monitor the trace of a request flow. Each trace message indicates a called microservice in the request flow. It can be formulated as $\langle Req, Caller, Callee, Duration \rangle$, where Req is the request ID, $Caller$ is the parent microservice that called the current microservice, $Callee$ is the current microservice, and $Duration$ is the execution time to finish the functionality of the current microservice. When a microservice suffers from performance issues, the execution duration on the microservice will be longer, thus the duration region of the corresponding trace message will show abnormal behavior. We leverage this feature, and report the suspicious microservice as long as the duration region of the corresponding trace messages continues to be much longer than before.

C. *Issue Identification and Localization*

We have got anomaly KPI queue from the KPI anomaly detection component, and suspicious microservices with logs from the trace/log analysis component. Based on these findings, we simplify the diagnosis process into three steps: issue identification, anomaly microservice localization and root cause indicators localization (as shown in Fig 2).

1) Issue Identification: We need to first identify if there is a performance issue, and then localize the root cause (if an issue really occurred). Motivated by real-world issue handling process, we extracted some features of the anomaly KPI queue based on its characteristics as follows:

- **Density of anomaly KPI queue** Number of KPI anomalies received per minute indicates the density of anomaly KPI queue. Empirically, more intensive anomalies are more likely to indicate a failure [27]. Therefore, we pick the density of anomalies as a feature.
- **Distribution of anomalies** Distribution of anomalies among microservices is highly correlated to the probability of failure occurring in different microservices. HotSpot [28] provided an algorithm to localize anomalies with multi-dimensional attributes with Monte Carlo Tree Search. In our scenario, the dimensionality of KPI is not that high, which makes it ineffective and too complicated to apply methods like HotSpot. So we simply calculate the standard deviation among the anomaly numbers of different microservices, and use it as a feature to describe the spatial distribution of anomalies. The denser the distribution, the more likely an issue is caused (by a single microservice or multiple ones).
- **Anomaly time span** Based on observation of real-world data, we find that most severe issues won't be automatically mitigated, that is, most anomalies will last until proper manual action has been taken. Intuitively, a longer consecutive anomaly queue suggests a higher possibility of an issue [29]. Therefore, we take the longest anomaly time span in the queue as a feature to represent the temporal distribution of anomalies.

In addition to the three features above, the suspicious microservices obtained from trace analysis is also an important feature, as it represents the most anomalous microservices from the perspective of the function call chain.

We design parameterized threshold-based rules to identify real issues. The thresholds of features may vary to adapt different microservice systems. Common threshold ranges of different features and parameters (including trace features and anomaly queue features) are shown in Table II.

TABLE II
COMMON THRESHOLD AND PARAMETER RANGES

Feature/Parameter	Range
Anomaly KPI Queue Density	2 to 30 (counts per minute)
Anomaly Time Span	120 to 600 (seconds)
Standard Deviation of Anomaly Counts	2 to 15 (counts)
Duration of Function Call	500 to 1000 (milliseconds)
Length of Sliding Window (T)	10 to 60 (points)
Length of Anomaly KPI Queue (K)	2 to 20 (anomalies)

An issue is recognized to have occurred only if all the rules are satisfied (i.e. all the thresholds are reached). For example, at a certain moment, there is 10 anomalies in the queue, the anomaly time span is 3 minutes, and the standard deviation of anomaly counts is 5.65. Supposing the density threshold is set to 8, time span threshold is 120 seconds and the standard deviation threshold is set to 7, then the current situation is not considered anomalous unless the standard deviation grows larger.

2) Anomaly Microservice Localization: After identifying an issue, we need to localize its root cause microservice and root cause indicators. Given anomaly KPI queue Q and suspicious microservice s obtained from trace analysis component, a voting mechanism is designed to localize root cause microservices first. Every anomaly in the queue Q votes for its corresponding microservice, and the suspicious microservice $SuspS$ gets α votes by default, where α is a parameter balancing the results from KPIs, traces and logs. The votes of each microservice $Votes(s_i)$ can be calculated as (3), where I is the indicator function.

$$Votes(s_i) = \alpha I(SuspS = s_i) + \sum_{j=1}^{len(Q)} I(q_j.s = s_i) \quad (3)$$

The microservice with most votes is regarded as the root cause microservice.

3) Root Cause Indicators Localization: To find root cause indicators, anomaly KPI queue is used to classify the issue into a certain subsystem (e.g. CPU subsystem, IO subsystem). The subsystem with the largest proportion of anomalous KPIs is considered to be responsible for the current issue.

After getting the anomalous subsystem, all KPIs from this subsystem will be double-checked by the KPI anomaly detection component with a lower anomaly score threshold. The double-check is to prevent missing some KPIs with relatively small change. The anomalous KPI result set can be denoted as $\{SuspKPI\}$.

Finally, if log analysis reported suspicious log entries (i.e. $\{SuspL\}$) from the root cause microservice, they will also be added to root cause indicators (denoted as RCI), which serve as a final output of PDiagnose.

$$RCI = \{SuspKPI\} \cup \{SuspL\} \quad (4)$$

V. EVALUATION

In this section, we aim to answer following research questions:

RQ1: How effective is PDiagnose for issue diagnosis?
RQ2: How efficient is PDiagnose?

A. Experimental Setup

1) Datasets: We evaluate PDiagnose on two open source datasets from International AIOps Challenge [30], which is a competition focusing on diagnosis of microservices systems. Here we use the 2020 and 2021 versions of the competition's open source datasets, which were collected from a large wireless provider and two commercial banks, respectively. The 2021 version of the dataset contains KPIs, traces and logs, while the 2020 version only contains KPIs and traces. KPIs are collected minutely, and each issue lasts for five minutes. The types of issues cover CPU, IO, Memory, Network and DB. The details of these datasets are shown in Table III. We believe these two datasets can serve as good examples of different real-world scenarios.

TABLE III
DATASET DETAILS

Dataset	Scenario	Microservices	Issues	Duration
AIOps Challenge 2020	wireless provider	61	81	14 days
AIOps Challenge 2021	commercial bank	35	19	2 days

2) Experiment Environment: All the experiments have been done on a server with eight 3.4GHz Xeon CPU cores and 16 GB memory. The algorithms are implemented in Python with popular libraries including scikit-learn [31], pandas [32], scipy [33] and xgboost [26]. To simulate a real-time diagnosis system, all the algorithms consume data from a Kafka [34] server continuously and submit diagnosis results as long as an issue is detected.

3) Parameters: The parameters are shown in Table IV.

TABLE IV
PARAMETER SETTINGS

Dataset	α	T	K	(w_1, w_2, w_3)
AIOps Challenge 2020	5	10	2	(0.5, 0.2, 0.3)
AIOps Challenge 2021	5	15	7	(0.7, 0.2, 0.1)

4) Measurement: we take recall scores $Recall$ to evaluate the performance of PDiagnose. Motivated by the fact that an undiscovered issue is often more crucial than a false positive, $Recall$ is chosen as it can better assess the comprehensiveness of a diagnosis approach than the top-k accuracy $AC@K$ used by MicroCause and FluxInfer. $Recall$ can be calculated as (5). I stands for the universal set of issues, $Result_i$ represents the diagnosis result (a set of indicators) of issue i, and RCI_i is the actual root cause indicator set of issue i.

$$Recall = \frac{|\{i|Result_i \supseteq RCI_i\}|}{|I|} \tag{5}$$

5) Baselines: We pick following baseline algorithms:

- **Random Selection** Random selection is a naive issue diagnosis method often taken by SRE teams with little experience. Trivially, operators manually check microservices selected randomly until the root cause microservice is found. After that, all the KPIs are checked one by one to diagnose the issue. Using it as a baseline, previous work [11] chooses different top-k accuracy $AC@K$ to test this method under different limitations. Because PDiagnose is not a learning-to-rank-based approach, in our case, we add a hard limitation (i.e. 20% of all microservices and KPIs) as maximum retries on this method to simulate a reasonable time limit.
- **MicroRCA** MicroRCA [11] is a dependency graph-based diagnosis approach. It utilizes service-level and system-level metrics to build anomaly graph and then uses PageRank algorithm to infer the root cause on the generated graph. From AIOps Challenge datasets, it is not possible to obtain the necessary data for building the system-specific anomaly graph. So we use PC algorithm to generate the graph instead.

B. Results

TABLE V
AVERAGED RECALL OF DIFFERENT ALGORITHMS

Dataset	Issue Type (Counts)	Random Selection	MicroRCA [11]	PDiagnose
2020	CPU (19)	0.235	**1.000**	**1.000**
	Network (50)	0.432	0.600	**0.820**
	DB (12)	0.429	0.000	**1.000**
	Total (81)	0.385	0.604	**0.889**
2021	CPU (4)	0.434	0.250	**1.000**
	IO (2)	0.269	0.000	**1.000**
	Memory (8)	0.293	0.000	**0.625**
	Network (4)	0.270	**0.500**	0.250
	Total (18)	0.317	0.167	**0.667**
Overall (99 Issues)		0.359	0.525	**0.848**

1) The Effectiveness of PDiagnose: To answer **RQ1**, we run those selected algorithms and PDiagnose with two datasets. The results are recorded in Table V, averaging 10 rounds of experiments. Note that the parameters of MicroRCA and PDiagnose are optimized manually.

From the results, we can observe that PDiagnose outperforms the two baselines with an overall recall of 0.848. Specifically, on two datasets, our approach outperforms MicroRCA by 28.5% and 50% respectively. MicroRCA handles network issues (50 out of 99 issues) quite well and performs better than Random Selection overall, while it fails to correctly diagnose IO, Memory and DB issues. Meanwhile, PDiagnose can diagnose all types of issues with satisfying recall.

For AIOps Challenge 2020 dataset, both PDiagnose and MicroRCA performed well dealing with CPU and network-related issues. However, only PDiagnose managed to correctly localize the 12 database-related issues. After a manual analysis on these issues, we find that database-related issues are usually reflected in just a few indicators within the database microservice, while many microservices' related indicators (e.g. CPU load) are affected. Dependency graph-based algorithms like MicroRCA leveraged causal mining algorithms to infer the relations among different indicators. Issues with larger scope will bring more vertexes in the dependency graph, and thereby result in worse accuracy. PDiagnose can better handle issues with large scope by the weighted voting mechanism, thus achieved better performance.

AIOps Challenge 2021 dataset contains more types of KPIs and root cause indicators, so all the three approaches got less satisfying recalls. Especially, the topological relationships among microservices and indicators are much more complicated, so MicroRCA performed poorly on this dataset (even worse than Random Selection), as a result of too many wrong causal edges. The anomaly KPI queue and voting mechanism leveraged by PDiagnose are application-agnostic, hence, the performance of PDiagnose was not affected significantly.

2) The Efficiency of PDiagnose: The efficiency of an issue diagnosis approach is a vital factor. SREs expect lower average diagnosis duration so the impact on the microservices and the loss of business can be minimized. To answer **RQ2**, we count the average diagnosis latency (time between the occurrence of an issue and the localization of it) of PDiagnose on AIOps Challenge 2021 dataset. Results have shown that it takes PDiagnose 157 seconds in average to correctly diagnose an issue. For minutely collected KPI time series, this result indicates that PDiagnose can diagnose an issue after only 2.6 data collecting intervals. In contrast, MicroRCA [11] and FluxInfer [9] need 4 and 10.6 data collecting intervals to correctly diagnosis an issue on average, respectively. This result shows the efficiency of PDiagnose is promising for microservices systems.

It is noteworthy that different approaches may have different data collecting intervals (5 seconds for FluxInfer, 0.25 seconds for MicroRCA), so it is not fair to directly compare the average diagnosis latency per issue of different approaches.

To further discuss the efficiency of different approaches, we can take a closer look at the complexity of building a dependency graph. As the most popular causal graph generation technique, PC algorithm is used by most of the existing works ([5]–[8], [12]). However, its worst case computational complexity is $O(p^q f(T))$ [13], where p is the dimensionality of variables (number of KPIs in our case), q is the maximal size of the neighborhoods (number of related microservices in our case) and $f(T)$ is the complexity of conditional independence test between two series with length T (depends on testing methods, often $O(T^2)$). As a microservices system grows, the exponential complexity of PC algorithm will soon be infeasible. For PDiagnose, as it consists only polynomial algorithms (The complexity of KDE is $O(T^3)$ [35], WMA has a complexity of $O(T)$ for series with length T, and issue identification step can be done with a maximum complexity of $O(pT)$), it has a polynomial complexity of $O(pT^3)$. In real world cases, p and q is proportional to the scale of microservices system, which can be very large (e.g. a system with 100 microservices and 100 metrics produces 10,000 series). The length of sliding window (T) is often smaller than p and q by a few orders of magnitude (shown in Table II and Table IV). Thus, PDiagnose is more efficient than graph-based approaches, especially for large-scale microservices systems.

VI. Threats to Validity

In this section, we discuss the threats to validity in our study as follows.

- **Different Types of Performance Issues** PDiagnose utilizes KPIs, logs and traces to diagnose performance issues. Although most of performance issues can manifest themselves via those indicators, there are still some types of latent issues that do not affect the indicators at all. For example, improper job orchestrating strategy among different CPU cores may cause performance issues in colocation-based microservices. As there are currently few indicators standing for microarchitecture-related root causes, this type of issue cannot be diagnosed by methods like PDiagnose.
- **Datasets, Target Systems and Measurements** To obtain real-world data from large-scale microservices systems (often hosted by large commercial companies) is quite challenging. Therefore, we use two open source datasets collected from wireless providers and commercial banks in our study. Of course, these two datasets cannot cover all the potential target systems, and we will deploy PDiagnose on more real-world systems to reduce this threat. Moreover, to evaluate PDiagnose, we took classic recall and diagnosis duration as metrics instead of widely used $AC@K$. Introducing more comprehensive metrics will be helpful for more solid and thorough evaluation.
- **Baselines** Picking baselines to evaluate PDiagnose is not easy. Most existing works focus on a specific scenario, so comparing the effectiveness of PDiagnose with these methods in a fair manner is infeasible. For example, FluxInfer [9] focuses on performance issues of **online database systems**, while MicroCause [10] diagnoses **a single** microservice instead of a holistic microservices system. We only choose Random Selection and MicroRCA as baselines to evaluate PDiagnose's effectiveness, because PDiagnose shares the most similar scenario with these two approaches. We believe more general-purpose issue diagnosis approaches will be proposed in the future, which will make the evaluation and comparison more useful and informative.

VII. Conclusion

We proposed PDiagnose, a performance issue diagnosis approach for microservices. Compared with existing approaches, by utilizing KPIs, logs and traces, PDiagnose can better understand the system's behavior and thus get better results. Moreover, PDiagnose does not rely on the expensive dependency graph-building phase to localize root causes, so its efficiency is promising for microservices systems.

In the future, we will combine PDiagnose with better log and trace anomaly detection approaches to further improve its performance.

Acknowledgment

Our greatest gratitude goes to the anonymous reviewers for their valuable suggestions, which improved this paper considerably. Also, we would like to thank Kangjin Wang, Yaoyong Dou, Kagami Tsukuyomi, Minghui Wang and all the members from PKU FineLab for their support. This work is supported by 2021 PKU-ZTE Cooperation Project.

References

[1] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of devops concepts and challenges," *ACM Comput. Surv.*, vol. 52, no. 6, Nov. 2019. I

[2] L. Chen, Y. Xu, Z. Lu, J. Wu, K. Gai, P. C. K. Hung, and M. Qiu, "Iot microservice deployment in edge-cloud hybrid environment using reinforcement learning," *IEEE Internet of Things Journal*, pp. 1–1, 2020. I

[3] K. Gai and M. Qiu, "Optimal resource allocation using reinforcement learning for iot content-centric services," *Applied Soft Computing*, vol. 70, pp. 12–21, 2018. I

[4] Z. Chen, "Very-large scale microservices architecture at practice: Alibaba's perspective," Alibaba Inc., Guangzhou, China, QCon Report, 2019. I

[5] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," in *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 93–104. I, II, V-B2

[6] 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. I, II, V-B2

[7] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2018, pp. 492–502. I, I, II, V-B2

[8] 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. Cham: Springer International Publishing, 2018, pp. 3–20. I, II, V-B2

[9] P. Liu, S. Zhang, Y. Sun, Y. Meng, J. Yang, and D. Pei, "Fluxinfer: Automatic diagnosis of performance anomaly for online database system," in *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*, 2020, pp. 1–8. I, II, V-B2, VI

[10] 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. I, II, VI

[11] 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*, 2020, pp. 1–9. I, I, II, V-A5, V-A5, V, V-B2

[12] M. Ma, W. Lin, D. Pan, and P. Wang, "Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications," in *2019 IEEE International Conference on Web Services (ICWS)*, 2019, pp. 60–67. I, II, V-B2

[13] M. Kalisch and P. Bühlmann, "Estimating high-dimensional directed acyclic graphs with the pc-algorithm," *J. Mach. Learn. Res.*, vol. 8, p. 613–636, May 2007. II, V-B2

[14] W. Xing and A. Ghorbani, "Weighted pagerank algorithm," in *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.*, 2004, pp. 305–314. II

[15] H. Nguyen, Y. Tan, and X. Gu, "Pal: Propagation-aware anomaly localization for cloud hosted distributed applications," in *Managing Large-Scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, ser. SLAML '11. New York, NY, USA: Association for Computing Machinery, 2011. II

[16] 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*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 19–33. II

[17] 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 *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 683–694. II

[18] T. Jia, L. Yang, P. Chen, Y. Li, F. Meng, and J. Xu, "Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, 2017, pp. 447–455. II

[19] S. Du and J. Cao, "Behavioral anomaly detection approach based on log monitoring," in *2015 International Conference on Behavioral, Economic and Socio-cultural Computing (BESC)*, 2015, pp. 188–194. II

[20] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue, and D. Pei, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 48–58. II

[21] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, Jul. 2009. IV-A

[22] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, vol. 54, no. 2, Mar. 2021. IV-A

[23] B. Silverman, *Density Estimation for Statistics and Data Analysis*. Routledge, 02 2018. IV-A

[24] R. Brown, *Smoothing, Forecasting and Prediction of Discrete Time Series*, ser. Dover Phoenix Editions. Dover Publications, 2004. IV-A

[25] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2019. IV-A

[26] "xgboost," https://xgboost.ai/, accessed: 2021-07-01. IV-A, V-A2

[27] N. Zhao, J. Chen, Z. Wang, X. Peng, G. Wang, Y. Wu, F. Zhou, Z. Feng, X. Nie, W. Zhang, K. Sui, and D. Pei, "Real-time incident prediction for online service systems," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 315–326. IV-C1

[28] Y. Sun, Y. Zhao, Y. Su, D. Liu, X. Nie, Y. Meng, S. Cheng, D. Pei, S. Zhang, X. Qu, and X. Guo, "Hotspot: Anomaly localization for additive kpis with multi-dimensional attributes," *IEEE Access*, vol. 6, pp. 10 909–10 923, 2018. IV-C1

[29] N. Zhao, P. Jin, L. Wang, X. Yang, R. Liu, W. Zhang, K. Sui, and D. Pei, "Automatically and adaptively identifying severe alerts for online service systems," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2420–2429. IV-C1

[30] "International aiops challenge," http://iops.ai, accessed: 2021-05-17. V-A1

[31] "scikit-learn," https://scikit-learn.org/, accessed: 2021-07-01. V-A2

[32] "pandas," https://pandas.pydata.org/, accessed: 2021-07-01. V-A2

[33] "Scipy," https://www.scipy.org/, accessed: 2021-07-01. V-A2

[34] "Apache kafka," https://kafka.apache.org/, accessed: 2021-07-01. V-A2

[35] S. Wang, J. Wang, and F.-l. Chung, "Kernel density estimation, kernel methods, and fast learning in large data sets," *IEEE Transactions on Cybernetics*, vol. 44, no. 1, pp. 1–20, 2014. V-B2