Diagnosing Performance Issues for Large-Scale Microservice Systems With Heterogeneous Graph

Lei Tao, Xianglin Lu, Shenglin Zhang, *Member, IEEE*, Jiaqi Luan, Yingke Li, Mingjie Li, Zeyan Li, Qingyang Yu, Hucheng Xie, Ruijie Xu, Chenyuan Hu, Canqun Yang, and Dan Pei, *Senior Member, IEEE*

***Abstract*—The availability of microservice systems is critical to business operations and corporate reputation. However, the dynamics and complexity of microservice systems introduce significant challenges to the performance issue diagnosis of large-scale microservice systems. After investigating hundreds of real-world performance issue cases in Tencent, we find that previous troubleshooting approaches fail to accurately localize root causes because they overlook the inconsistency between causality and calling relationships. Therefore, we propose a novel approach, MicroDig, to diagnose performance issues for large-scale microservice systems. Specifically, MicroDig constructs a heterogeneous propagation graph to capture the causal relationships between calls and microservices. It then conducts a heterogeneity-oriented random walk (HORW) to pinpoint the culprit microservice. Extensive evaluation experiments have been conducted to evaluate MicroDig's performance on 60 real-world performance issues collected from Tencent, 80 manually injected ones collected from a widely used open-source microservice system and 128 performance issues collected from an e-commerce system used by a top-tier global commercial bank. MicroDig achieves 94.1%, 85.5% and 93.8% top-3 accuracy on the three datasets, respectively, significantly outperforming six popular baseline methods. Additionally, we have shared our success stories and learned lessons from the deployment of MicroDig in Tencent.**

***Index Terms*—Heterogeneous propagation graph, microservice systems, performance issue diagnosis.**

Manuscript received 25 October 2023; revised 23 March 2024; accepted 27 April 2024. Date of publication 17 May 2024; date of current version 9 October 2024. This work was supported in part by the Advanced Research Project of China under Grant 31511010501 and in part by the National Natural Science Foundation of China under Grant 62272249 and Grant 62072264. *(Lei Tao and Xianglin Lu contributed equally to this work.) (Corresponding author: Shenglin Zhang.)*

Lei Tao, Jiaqi Luan, and Yingke Li are with Nankai University, Tianjin 300192, China (e-mail: leitao@mail.nankai.edu.cn; jiaqiluan@mail.nankai.edu.cn; yingkeli@mail.nankai.edu.cn).

Xianglin Lu, Mingjie Li, Zeyan Li, Qingyang Yu, and Dan Pei are with Tsinghua University, Beijing 100190, China, and also with Beijing National Research Center for Information Science and Technology, Beijing 100084, China (e-mail: luxl20@mails.tsinghua.edu.cn; lmj18@mails.tsinghua.edu.cn; zy-li18@mails.tsinghua.edu.cn; yqy17@mails.tsinghua.edu.cn; peidan@tsinghua.edu.cn).

Shenglin Zhang is with the College of Software, Haihe Laboratory of Information Technology Application Innovation, Nankai University, Tianjin 300192, China, and also with the Tianjin Key Laboratory of Software Experience and Human Computer Interaction, Tianjin 300192, China (e-mail: zhangsl@nankai.edu.cn).

Chenyuan Hu is with the National Supercomputing Center of Tianjin, Tianjin 300456, China (e-mail: cheneyhu@tencent.com).

Hucheng Xie, Ruijie Xu, and Canqun Yang are with Tencent, Inc., Beijing 100080, China (e-mail: toraxie@tencent.com; rjxu@tencent.com; canqun@nudt.edu.cn).

Digital Object Identifier 10.1109/TSC.2024.3402172

I. Introduction

MICROSERVICE architecture is a scheme to develop a single application into a set of small services [1]. Each microservice is developed and runs independently and communicates with each other through lightweight communication mechanisms. However, with the rapid evolution and scale expansion of microservice systems, reliability, and availability maintenance are challenging due to the inherent dynamics and complexity [2], [3], [4]. Nevertheless, powerful system performance and high-quality user experience are critical to underpinning the reputation and profitability of the enterprise, otherwise may cause significant losses. For example, the estimated cost of Amazon's downtime for an hour during the most prominent promotional event is as high as 100 million dollars [5].

Therefore, operators make great efforts to maintain system performance. They monitor system health by configuring and collecting SLIs (Service Level Indicators) such as QPS (Queries Per Second), response time, and success rate. They also configure the corresponding SLOs (Service Level Objects) for user-facing microservices to evaluate system performance, e.g., keeping response time under 10 ms while handling 1000 QPS [6]. When the status of a user-facing service fails to meet the predefined SLO, the microservice system is considered anomalous, and a performance issue will be generated. Since different operation teams usually manage different services, it is necessary to localize the culprit microservice when a performance issue happens, so as to assign the performance issue ticket to the right operation team. However, a microservice system consists of a large number of services (In this paper, we use "microservice" and "service" interchangeably). For example, the e-commerce system of Alibaba contains more than 30,000 services [2]. In addition, the internal relationships of the services are dynamic and complex [7], [8], [9]. Anomalies can propagate among services, causing availability issues of different services simultaneously. Thus, manual localization is laborious and time-consuming. According to IBM's statistics [10], root cause localization takes the longest time in the entire performance issue handling period, which needs to be optimized urgently. Therefore, our work focuses on efficiently localizing the culprit (i.e., the root cause service) when a performance issue (i.e., an SLO violation) occurs in the microservice system.

Some existing works [11], [12], [13] proposed to localize the root cause service via trace analysis. The service-level traces

are collected from the distributed tracing framework, which records the complete service invocation process of each request execution in the microservice system. However, with the increasing number of microservices and requests, the storage requirements also tremendously increase, which brings about significant overheads. For example, in eBay, the microservice systems produce nearly 150 billion traces per day [14]. Therefore, more and more enterprises, including Tencent, choose to retain the end-to-end aggregated calls between every two services instead of the entire traces.

Some works are carried out on the aggregated call data. The method proposed in [15] recommends the most similar historical anomalies through pattern matching for root cause localization. However, it depends heavily on sufficient samples of historical performance issues and the high coverage of issue types, leaving it impractical. Other works employ causal graph-based methods for root cause localization. However, calling relationships (used in [2], [7], [16]) are insufficient to build causal graphs (see Section IV for more details), while causality mining algorithms (used in [17], [18]) suffer from high computational cost and low accuracy when the number of microservices is enormous. Therefore, a more practical root-cause localization approach for microservice systems is urgently needed.

In this paper, we propose an accurate and efficient method, *MicroDig*, to localize the root cause for large-scale microservice systems. The foundation of *MicroDig* is that dynamic calling relationships are not equivalent to causality, which will be further elaborated on in Section IV. When a performance issue occurs in *MicroDig*, we first identify its association calls to avoid the interference of irrelevant microservices. Then, since the relationship between microservices is complex and dynamic, we build a causal graph based on the calling relationships near the occurrence time of the performance issue. To address the inconsistency between causality and calling relationships, we apply both microservices and their calls as nodes to build a heterogeneous propagation graph through causal relationships, whose edges represent causal relationships. Usually, a ranking algorithm is used on a causal graph to get the root cause. However, traditional ranking algorithms cannot be directly applied to heterogeneous causal graphs. Thus, we propose a novel ranking algorithm, Heterogeneity-Oriented Random Walk (HORW), combining correlation coefficient and anomaly detection results for candidate culprits ranking.

Extensive evaluation experiments have been conducted to evaluate the performance of *MicroDig* using 60 real-world performance issues collected from Tencent, a top-tier global multimedia service provider serving over 1 billion daily active users, 80 manually injected performance issues collected from a widely used open-source microservice system Train-Ticket [19] and 128 performance issues collected from an e-commerce system used by a top-tier global commercial bank. Experimental results show that *MicroDig* ranks the root-cause microservices at the top 3 in 94.1%, 85.5% and 93.8% performance issues on the real-world and simulated datasets, respectively, significantly outperforming six popular baseline methods by 16.0%, 32.2% and 88.5%. Furthermore, we also discuss the success stories and lessons learned from the deployment of *MicroDig* in Tencent. To facilitate follow-up studies, our code is released at [20].

To sum up, our work has the following main contributions:

- After investigating hundreds of performance issue cases of Tencent, we identified a valuable problem, i.e., the inconsistency of causality and calling relationships.
- We propose *MicroDig* to localize the root cause of performance issues for large-scale microservice systems. Its core technologies include the construction of the heterogeneous graph and HORW. The former aims to accurately model causality based on calling relationships. The latter is designed to rank the suspicious services on the heterogeneous graph to find the culprit.
- We deploy the prototype of *MicroDig* in Tencent. The deployment demonstrates the superior performance of *MicroDig* in root cause localization for large-scale microservice systems.

II. Related Work

Some work utilizes microservice invocation traces for root cause localization. TraceRank [21] calculates the suspiciousness of each service based on the anomalous traces associated with the service, and conducts random walk on the service call graph to pinpoint anomalous services. TraceAnomaly [11] first performs anomaly detection on traces based on Variational Autoencoder (VAE) and then identifies the service corresponding to the longest anomalous call as the root cause. TraceRCA [12] applies a unified indicator to measure the possibility of service becoming the root cause according to the number of anomalous traces that pass through the service. Sage [13] builds an impact graph between services based on domain knowledge and applies deep learning algorithms to perform counterfactual reasoning on the graph. However, with the dramatic increase in the number of services, storing data in the form of a trace is prohibitively expensive. Therefore, the above methods are inappropriate for Tencent's large-scale microservice systems, and we did not compare them in the evaluation experiments.

Some researchers work on the aggregated invocation data of pairwise services. We divide the related work into the following categories. The first category is based on fault similarity matching. For example, the method proposed in [15] localizes the root cause service by calculating the similarity between the new fault graph and historical fault graphs and recommends the most similar fault graphs labeled by experts. Although this type of method is unsupervised, it still needs enough historical fault samples of different types for pattern matching which is hard to obtain in practice. Therefore, this type of method is impractical, and it is not considered in our comparative evaluation.

Other localization methods consist of two key steps: causal graph construction and ranking on the graph. In terms of graph construction, some methods utilize the calling relationship to construct. FChain [22] pinpoints the culprit component by analyzing the anomalous change points of different components and the invocation topology, which catches the inter-component dependency information. MonitorRank [16] achieves ranking by implementing the personalized PageRank algorithm on the call

graph. MicroRCA [7] constructs an anomaly graph based on the calling relationship and deployment relationship between services and employs Personalized Pagerank to localize the root cause service on the anomaly subgraph. It heavily relies on the information of real-time service deployment, which is dynamically changing and cannot be easily obtained in our scenario. Therefore, we did not take it as a baseline method in our evaluation experiments. MicroHECL [2] first detects service anomalies and then analyzes possible anomaly propagation paths according to different types of fault propagation modes on the call graph, and ranks candidate root causes through correlation analysis. However, calling relationship and causality are not completely equivalent as mentioned in Section IV so that it is inaccurate for such methods to construct causal relationships directly by calling relationships. Furthermore, some methods employ causality mining algorithms to build causal graphs and perform localization. MS-Rank [23], CloudRanger [3], and ServiceRank [17] utilize conditional independence tests for mining dynamic causal relationships between services for impact graph building and rank the services on the graph based on the random walk. Specifically, ServiceRank constructs the impact graph based on PC algorithm and identifies the root cause based on the second-order random walk. Nevertheless, when the number of services in a microservice system is enormous, using causality mining algorithms will be less efficient and accurate. After construction, most of the above methods apply random walk or pagerank algorithms on the graph to get the root cause, and also some methods determine the root cause by searching directly on the causal graph. CauseInfer [18] collects connection information to build a service dependency graph, uses depth-first search to find the leaf nodes along the anomalous path, and calculates the anomaly score based on Z-Score to get the final ranking. Microscope [24] constructs the causal graph with calling relationships and causal mining algorithms, searches all boundary nodes as candidate services along the reverse direction, and ranks them according to their similarity with the performance metric. It is difficult for such methods to guarantee the effectiveness of root cause localization when the causality mining algorithm or anomaly detection algorithm is inaccurate or untrustworthy.

Considering the practicability of the algorithm and the fairness of the evaluation, we choose the latest or most influential algorithm for each type of method as the baseline comparison in the experimental part.

III. Background

A. Microservice System

A microservice system decomposes large applications into multiple independent microservices, each with its area of responsibility. It is designed to handle discrete tasks and enables independent development, deployment, and maintenance of services. When processing a user request, a microservice-based application may invoke many internal services to collectively generate its response to provide business support for the application.

The microservices in the system are loosely coupled. They communicate with each other through simple interfaces or protocols such as RESTful (Representational State Transfer) APIs (Application Programming Interfaces) and RPCs (Remote Function Calls). In general, the communication relationship of microservices presents a mesh structure. Thus, we model the communications as a graph, where the vertices represent the microservices and the edges point from the caller to the callee providing the service. An example of the call process is shown in Fig. 1, which depicts a basic synchronous service request. The example involves three microservices, A, B, and C, whose relationships are represented with arrows, i.e., A calls B (denoted as $A \rightarrow^{call} B$), and B calls C (denoted as $B \rightarrow^{call} C$). The processing logic of service A consists of three parts. The first is the pre-call logic, and then the downstream service is called. After the call returns, the post-call logic is executed. So do services B and C.

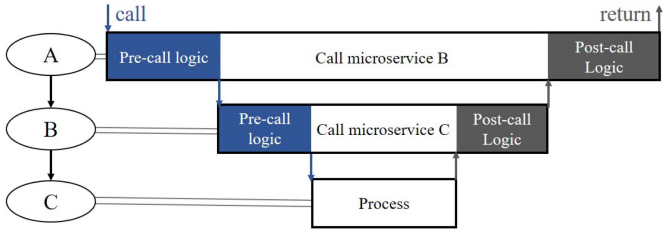
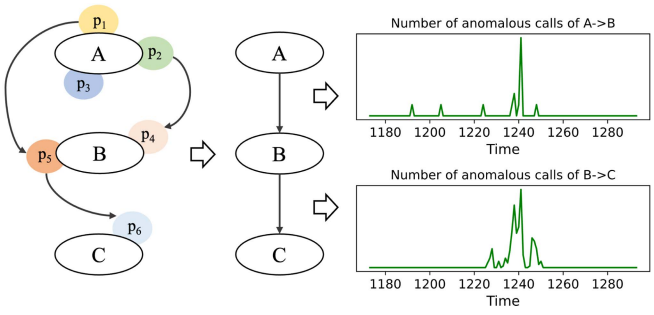


Fig. 1. A toy example of the call process in the microservice system.

The anomaly of a particular service in the microservice system can propagate to and affect other services. The main reason is that the return result of the upstream call is determined by that of the downstream call. For example, in the case of Fig. 1, when the performance of service C degrades, it will cause an increase in response time, the duration of $B \rightarrow^{call} C$ increases, and so does the duration of $A \rightarrow^{call} B$. Similarly, if service C cannot handle B's request due to overloading and returns an exception, it usually results in an exception of $A \rightarrow^{call} B$. This does not mean that the downstream call will cause the anomaly of the upstream call in every performance issue, but the anomaly of the upstream call may be caused by the anomaly of the downstream call.

B. Performance Monitoring

1) Data Record: There are many solutions for distributed tracing of service calls, such as Jaeger [25], Zipkin [26], and OpenTelemetry [27]. However, as the number of microservices increases, the cost of storing all traces becomes unbearable. To mitigate the storage pressure, only storing aggregated end-to-end call data has become a better choice for some enterprises, including Tencent. For end-to-end pairwise call records, multiple call fields are reserved within each predefined period (e.g., 1 minute). The most basic collection items are the number of anomalous calls, including exception calls and timeout calls, and the total number of calls. In the call information retained by Tencent, in addition to the service names of the caller (client) and callee (server), the corresponding ports are also recorded. One microservice can communicate with others through different ports which are responsible for different types of functions, e.g., functions supporting different protocols. For example, as



(a) Calls between microservices (b) A real performance issue case

Fig. 2. An example of port-level calls and the related performance issue.

shown in Fig. 2(a), Service A and B have three and two different ports, respectively. Service A calls Service B via $p_2 \rightarrow^{call} p_4$ and $p_1 \rightarrow^{call} p_5$. This type of port-level call data collected by the monitoring system in Tencent is used in the remaining parts.

2) Performance Issue Diagnosis: In a real-world production environment, the primary performance concern of a microservice is its high latency and low availability, which can be perceived from the number of timeout calls and exception calls, respectively. Performance issues can arise for various reasons, e.g., limited host resources, hardware availability, and network instability. When a service has performance issues, the anomalies will propagate along the calling topology, which may eventually affect the external services provided by the microservice system. To guarantee superior user experience, operators configure SLIs and SLOs to monitor the availability of microservice systems. When a service running state fails to meet the SLO standard, a performance issue will be detected. Due to the large number of services, complex and dynamic relationships, and the inconsistency between the calling relationship and the causal relationship in the microservice system, determining the root cause remains challenging.

IV. Motivation

A. Motivating Example

Some works take the calling relationship as the causality of anomaly propagation [2], [7], [16], which is oversimplified in our scenario. Fig. 2(b) shows a real performance issue where A, B, and C represent three microservices, respectively.

The number of anomalous calls of both $A \rightarrow^{call} B$ and $B \rightarrow^{call} C$ increase as shown in Fig. 2. However, as operators dug into the details, they found no meaningful error reports of C. On the other hand, those calls from B (i.e., $B \rightarrow^{call} C$) are anomalous. This is because B had exhausted the file descriptors and failed to set up new connections to C. As calling data is recorded by callers in Tencent, such anomalies contributed to the anomaly rate of $B \rightarrow^{call} C$ as well. In conclusion, *the anomaly rates of* $A \rightarrow^{call} B$ *and* $B \rightarrow^{call} C$ *are confounded by the system resources of* B [28]. Hence, B is the root cause service in this case, which cannot be localized by searching along the calling relationship. After investigating hundreds of performance issues of Tencent, we found that over 35% of performance issues suffer from such problems.

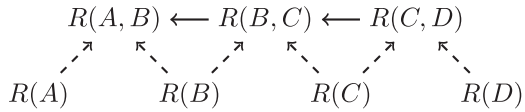


Fig. 3. The graphical model of $A \rightarrow^{call} B \rightarrow^{call} C \rightarrow^{call} D$. $R(A, B)$ represents the anomaly rate of $A \rightarrow^{call} B$ and $R(A)$ is the anomaly rate of the service A itself.

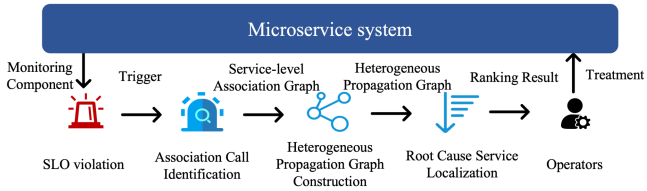


Fig. 4. The framework of *MicroDig*.

B. Heterogeneous Propagation Graph

Based on the above observations, we found that a call's anomaly cannot be just directly attributed to the downstream service. Instead, both the caller and the callee can contribute to this situation. Therefore, it is not sufficient to capture the anomaly propagation only with the calling relationship. In this paper, we propose a graphical model [28], i.e., a *heterogeneous propagation graph*, to describe the causal relationship more precisely than the calling relationship. As shown in Fig. 3, a heterogeneous propagation graph considers both observed variables for each call (e.g., the anomaly rate $R(A, B)$ of $A \rightarrow^{call} B$) and unobserved ones for services (e.g., the anomaly rate $R(A)$ of the service A itself).

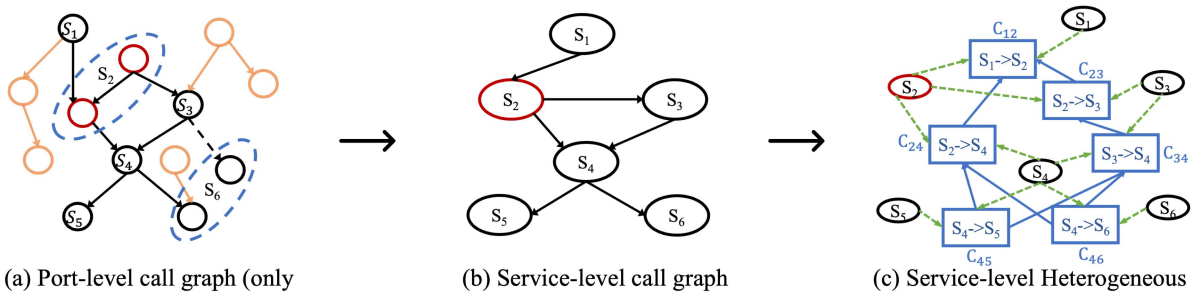
Based on the call and return mechanism of the microservice system, we can easily identify the causal relationship between upstream and downstream calls. Specifically, $R(A, B)$ describes all the possible anomalies after the invocation starts, which consist of those in B (i.e., $R(B)$) or the subsequent callings from B. Besides, the motivating example mentioned in Section IV-A illustrates that such an anomaly can also arise in A. Hence, we propose to link all services to related calls (i.e., the dashed arrows in Fig. 3).

The missing edges in Fig. 3 encode our assumptions. As the anomaly rates of services are unobserved in this work, we assume that services are deployed independently to simplify our model, e.g., $R(A)$ and $R(B)$ are independent. Furthermore, we assume that two calls without a joint node are not directly related, e.g., there is no arrow from $R(C, D)$ to $R(A, B)$ in Fig. 3. We leave a complete model for future work.

V. *MicroDig*

A. Overview

Fig. 4 shows the core framework of *MicroDig*. When the monitoring component of the microservice system detects an SLO violation, *MicroDig* is triggered to localize the culprit. Three key phases make up *MicroDig*. First, in the association calls identification phase, *MicroDig* search from the issue



(a) Port-level call graph (only anomalous and associated calls)

(b) Service-level call graph (aggregated from the previous graph)

(c) Service-level Heterogeneous Propagation Graph

Fig. 5. The construction process of heterogeneous propagation graph.

microservice to find candidate microservices associated with the performance issue (Section V-B). Second, a heterogeneous propagation graph is constructed with inter-microservice calls and candidate microservices as nodes in the heterogeneous propagation graph construction phase (Section V-C). Third, in the phase of culprit service localization, a heterogeneity-oriented random walk algorithm is proposed to perform microservice ranking on the heterogeneous graph (Section V-D).

B. Association Call Identification

Large-scale microservice systems usually generate plenty of service calls in a short time. As mentioned in Section III-B, only core fields, including the number of calls, exception calls, and timeout calls per minute, are stored. When a performance issue (i.e., an SLO violation) is detected, there are lots of calls near the issue. Analyzing all recorded calls is not only inefficient but also leads to poor root-cause localization accuracy. The main reason is that a large number of service calls are irrelevant to the performance issue, so we need to identify related microservice calls.

As discussed in Section III-B, the data provided by Tencent contains the calling port of each microservice (i.e., on the port level), but our objective is to select the culprit microservice (i.e., on the service level). In order to more accurately ascertain the microservices associated with the performance issue, we first choose to build a calling graph of port level. Then we retain related calls and corresponding microservices by performing BFS (Breadth-First Search) and anomaly detection on the graph. As shown in Fig. 5(a), a circle represents a port-level node, the ports in the same dotted ellipse belong to the same microservice, the orange port-level nodes are issue-irrelevant, and calls denoted by dashed lines denote non-anomalous ones. Finally, the port-level nodes in the graph are aggregated to the service level, as shown in Fig. 5(b). We detect anomalous ports first because if we perform port aggregation first, an anomalous port-level call may be overwhelmed by the normal calls of the same microservice, which may cause the root cause services to be filtered out at this step.

1) Construction of Association Graph: Since the recorded data encode the calling relationship among microservice ports, a call graph with service ports as nodes can be easily constructed. Then, we consider all port-level nodes on the call graph that belong to the microservice may suffer from the performance issue. Starting from these nodes, two BFSes are performed along and against the direction of the calling edges. The sub-graph consisting of all traversed port-level nodes associated with the issue service is the port-level association graph.

To capture the propagation pattern of the performance issue and further, narrow down the number of candidate root causes, we perform anomaly detection using an efficient and widely used anomaly detection method (called *k-sigma*) for each edge (i.e., call) in the association graph. It learns parameters μ and σ from historical data and treats the value exceeding $(\mu - k * \sigma, \mu + k * \sigma)$ as anomalies. In this work, we take k as 3, because the system in Tencent also has some fluctuations in the normal state.

We consider a call anomalous when the value of the call's exception rate or timeout rate is detected as anomalous. Given an issue starting at time point t, we use each call's exception rate or timeout rate from $t - \phi$ to $t + \psi$ for anomaly detection. Data within $[t - \phi - \delta, t - \phi)$ are used to learn the parameters of the normal pattern.

2) Data Merge: We conduct anomaly detection based on the port-level data instead of the service-level data because an anomaly at the port level, which indicates that the service containing the port experiences anomalous behavior, may be overwhelmed if we conduct anomaly detection based on the service-level data aggregated from the port-level data. Based on the above steps, edges with anomalies in the association graph are retained. Since the graph we get is a port-level association graph, and our objective is to localize the culprit service, we need to merge the call data of ports and construct the service-level association graph.

Let p be a node of the port-level association graph. We denote all the ports of a given service S as $P(S)$. To construct the service level graph, we merge all the port-level nodes of the same service S to one node denoted as $S = \{p \in P(S)\}$. The anomaly rate $R(S, S')$ of an edge $S \rightarrow^{call} S'$ at the service level integrates the number of anomalous calls $F(p, p')$ and that of the total calls $N(p, p')$ for each related port-level edge $p \rightarrow^{call} p'$. For time point t, $R_t(S, S')$ is calculated as:

$$R_t(S, S') = \frac{\sum_{p \in S, p' \in S'} F_t(p, p')}{\sum_{p \in S, p' \in S'} N_t(p, p')} \tag{1}$$

Finally, the association graph at the service level whose nodes are all relevant to the issue service is constructed. Meanwhile, we can obtain a time series of anomaly rates for $S \rightarrow^{call} S'$: $R(S, S') = (R_{t-\phi}, R_{t-\phi+1}, \ldots, R_{t+\psi})$.

C. Heterogeneous Propagation Graph Construction

The association graph constructed is directed, and the direction of its edges represents the calling relationship between services. However, as mentioned in Section IV, the calling relationship is not equivalent to the causal relationship. Therefore, the association graph cannot be directly used as a causal graph for root cause localization. We propose to build a heterogeneous propagation graph that reflects the causal relationship between the calls and the services based on the association graph.

As shown in Fig. 5(c) and line 2 and line 10 in Algorithm 1, we first abstract the edges in the association graph (i.e., edge $S_1 \rightarrow S_2$, edge $S_2 \rightarrow S_3$, ...) as *call nodes* (denoted as $C_{12}, C_{23}, \ldots$) and the nodes in the association graph as *service nodes* (denoted as $S_1, S_2, \ldots$). Then, we add edges between these nodes in the causal graph as follows.

The edge between a call node and a service node: Since the caller and callee of one service call are the immediate cause of the call, we add two edges from both the caller and callee service nodes to each call node, respectively, as shown by the green dashed edges in Fig. 5(c) and line 11-15 in Algorithm 1.

The edge between two different call nodes: Our key insight is that the anomaly of the downstream call can cause the anomaly of the upstream call. Therefore, for adjacent calls on the service call chain, we argue that the downstream call is the direct cause of the upstream call and we thus add an edge from the downstream call node to the upstream call node, as shown by the solid blue edges in Fig. 5(c). For instance, as shown in Fig. 5(b) and line 3-9 in Algorithm 1, edge $S_2 \rightarrow S_4$ and edge $S_2 \rightarrow S_3$ are direct downstream calls of edge $S_1 \rightarrow S_2$ in the association graph, so an edge from the call node C_{24} to the call node C_{12} and an edge from the call node C_{23} to the call node C_{12} are added to the propagation graph, respectively.

Since the calling relationships and their corresponding metric data do not reflect the causal relationship between service nodes, we do not add edges between different service nodes in the propagation graph. Finally, a heterogeneous propagation graph with different types of nodes and edges is constructed.

Note that in a propagation graph, the edges point from the causes to the effects. To perform root cause localization, we flip every edge in the propagation graph and make the edges point from the effects to their causes.

Algorithm 1: Heterogeneous Propagation Graph Construction.

Input: G: association graph at service level.
Output: G_h: Heterogeneous propagation graph.

```
1: G_h ← initialize the heterogeneous propagation graph
2: G_h.addNodes(G.edges())
3: for all S ∈ G.nodes() do
4:   for all C_out ∈ G.outEdges(S) do
5:     for all C_in ∈ G.inEdges(S) do
6:       G_h.addEdge(C_out, C_in)
7:     end for
8:   end for
9: end for
10: G_h.addNodes(G.nodes())
11: for all C ∈ G.edges() do
12:   caller, callee ← splitCall(C)
13:   G_h.addEdge(caller, C)
14:   G_h.addEdge(callee, C)
15: end for
```

D. Root Cause Service Localization

To localize the culprit (root cause) service in the heterogeneous propagation graph, we propose a novel method called Heterogeneity-Oriented Random Walk (HORW) which fully considers the characteristics of heterogeneity, and innovates in the calculation of transition probability.

1) Transition Weight Calculation: There are two types of edges in the heterogeneous propagation graph: (1) edges that connect two different call nodes and (2) those that connect one call node and one service node.

For the first type of edges, suppose the start and end nodes of an edge are C_{12} and C_{23}, respectively. Since we have already obtained the time series data of the anomaly rates of C_{12} and C_{23}, i.e., $\mathbb{R}(S_1, S_2)$ and $\mathbb{R}(S_2, S_3)$, respectively, we apply the correlation coefficient of $\mathbb{R}(S_1, S_2)$ and $\mathbb{R}(S_2, S_3)$ as the weight of the edge from C_{12} to C_{23}.

For the second type of edges, it seems impossible to determine which service node has a larger causal influence on a given call node since we only have monitoring data of service calls. To address this problem, we use all service call information related to a service node (not only the node on the service-level association graph) to calculate the anomaly score of the service node, and assign different transition wights for the edges from the calling node to the caller/callee service node according to their service anomaly scores. The details are described as follows.

For a given service S, we denote its upstream service set as $\mathbb{S}_U = \{S' \mid S' \rightarrow^{call} S\}$ and its downstream service set as $\mathbb{S}_D = \{S' \mid S \rightarrow^{call} S'\}$. Let $\theta(S')$ be the anomaly indicator of S'. When any related port-level call between S and S' is detected as anomalous, $\theta(S') = 1$. Otherwise, $\theta(S') = 0$. Finally, the anomaly score of the given service S, α_S measures the anomalous ratio of its related services as:

$$\alpha_S = \frac{|\{S' \mid S' \in \mathbb{S}_U \cup \mathbb{S}_D, \theta(S') = 1\}|}{|\mathbb{S}_U \cup \mathbb{S}_D|} \tag{2}$$

Given a call node C connecting two service nodes S_{caller} and S_{callee} in the heterogeneous propagation graph, let ω_{caller} be the weight of the edge from C to S_{caller} and ω_{callee} be the weight of the edge from C to S_{callee}. Recall that we have obtained the weights between different call nodes according to the first type of edges. We sum the weights of all the out (in) edges between C and its associated call nodes, denoted as η_{out} (η_{in}). Intuitively, we will partition $\Delta_\eta = \eta_{in} - \eta_{out}$ between ω_{caller}

Algorithm 2: Heterogeneity-Oriented Random Walk.

Input: G: weighted heterogeneous propagation graph with M call nodes and L service nodes, $\mathbf{T}$: transition probability matrix of G, $\varepsilon(C)$: the initial score of call node C, and κ: the number of iterations.
Output: $\mathbf{P}$: The root cause probability of each node.

1: **for all** call nodes C in G **do**
2: $\quad \mathbf{A}(C) \leftarrow \varepsilon(C) / \sum_{i=1}^{M} \varepsilon(C_i)$
3: **end for**
4: $\mathbf{V} \leftarrow \mathbf{0}$ {the number of times each node is visited}
5: **for** $i = 1$ to κ **do**
6: $\quad n \leftarrow$ Randomly pick a call node according to $\mathbf{A}$
7: $\quad$ **for** $j = 1$ to $M + L$ **do**
8: $\quad\quad n \leftarrow$ Randomly pick the next node according to $\mathbf{T}(n)$
9: $\quad\quad \mathbf{V}(n) \leftarrow \mathbf{V}(n) + 1$
10: $\quad$ **end for**
11: **end for**
12: $\mathbf{P} = \mathbf{V} / \sum_{j=1}^{M+L} \mathbf{V}_j$

and ω_{callee} equally when $\Delta_\alpha = \alpha(S_{caller}) - \alpha(S_{callee}) = 0$ and more (less) for ω_{caller} when $\Delta_\alpha > 0$ ($\Delta_\alpha < 0$). β is a preset hyperparameter, which is related to the accuracy of anomaly service detection. It can well mitigate the impact of wrong anomaly. Finally, ω_{caller} and ω_{callee} are calculated using (3), where $\operatorname{sgn} x$ is $-1, 0, 1$ when $x < 0, x = 0, x > 0$, respectively.

$$\begin{aligned} \omega_{caller} &= \max(0, \Delta_\eta) * [0.5 + \beta \operatorname{sgn}(\Delta_\alpha)] \\ \omega_{callee} &= \max(0, \Delta_\eta) * [0.5 - \beta \operatorname{sgn}(\Delta_\alpha)] \end{aligned} \tag{3}$$

2) Heterogeneity-Oriented Random Walk (HORW): We now get an initial weighted heterogeneous graph. But it cannot be directly used for root cause localization yet, because a native random walker may be stuck in a node without out-edges, no matter how irrelevant those nodes are to the performance issue. Therefore, we add an additional backward edge to each pair of nodes linked by only one directed edge in the heterogeneous propagation graph, whose weight is ρ times the weight of the original edge, so that the random walker can flexibly explore the nodes with high pattern similarity. We also add a self-loop edge to each service node in the graph, whose weight is the max of all the in-edge weights of the node. After all the steps above, the weighted heterogeneous graph is finally constructed, and we can get a transition probability matrix $\mathbf{T}$ by normalizing the out-edge weights of each node. Then, we propose Algorithm 2 to rank the root cause services.

Before sorting, we take the call nodes related to the issue service as the random walk entry nodes. The initial scores of these nodes are set to 1, and the others are set to 0. In Algorithm 2, we first normalize the initial score of each call node to their initialization probability (line 1-3). Then, we apply HORW on graph G and get the final probability vector $\mathbf{P}$ (line 4-12). Finally, we can sort all the nodes in the heterogeneous graph according to $\mathbf{P}$. We filter out the call nodes in the sorting results, and the remaining is the root cause ranking result of the candidate service nodes.

VI. Experiment

We conduct a variety of experimental studies to answer the following research questions.

- *RQ1:* How accurate is *MicroDig* in performance issue diagnosis?
- *RQ2:* Can *MicroDig* efficiently diagnose a performance issue for large-scale microservice systems?
- *RQ3:* Whether the core components of *MicroDig* significantly contribute to the performance of *MicroDig*?

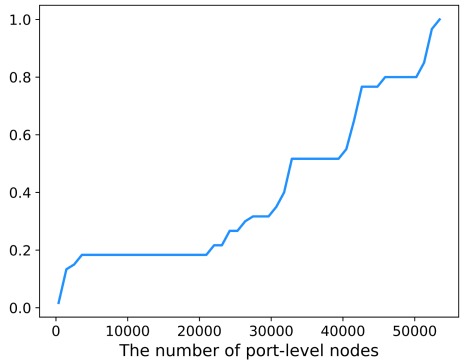
A. Dataset

We evaluate the performance of *MicroDig* using 60 real-world performance issues collected from Tencent, a top-tier global multimedia service provider housing services for over 1 billion daily active users, 80 manually injected ones collected from Train-Ticket [19], a widely used open-source microservice system and 128 performance issues collected from an e-commerce system used by a large bank. To help readers further extend this work in the future, we release our code at [20].

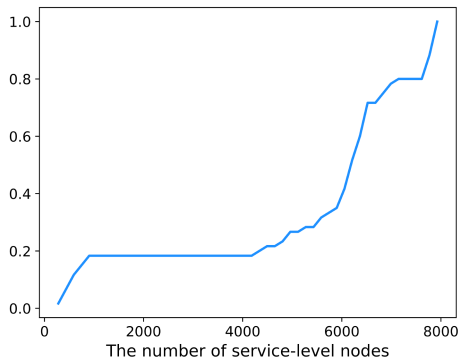
1) Real-World Production Microservice System: The operators of Tencent have configured SLOs for important user-facing services. We collected 60 real-world performance issues according to SLO violations in a subsystem of Tencent's video business with more than 8,000 microservices. Each case was examined by professional operators of Tencent to point out the root cause. The time span of the data collection ranged from December 2021 to July 2022. The data recorded the number of anomalous calls (including timeout calls and exception calls) and the total calls per minute. The cumulative distribution function (CDF) of the number of port-level and service-level nodes are shown in Fig. 6. We can observe that 80% of performance issues associated with 20,000+ port-level nodes and 4,000+ service-level nodes. We denote the real-world dataset of Tencent as $\mathcal{A}$ hereinafter.

2) Open-Source Microservice System: Train-Ticket [19] is a Web-based online ticket booking microservice system, containing 41 microservices. It has been widely used in previous works [11], [12], [19]. We used Kubernetes [29] for container orchestration and management on 7 physical machines. We injected 6 types of performance issues (i.e., network loss, network corruption, network delay, memory stress, CPU stress, and pod failure). Each performance issue lasted for about five minutes. We collected a total of 80 performance issues injected on different services. We denote the dataset collected from Train-Ticket as $\mathcal{B}$ hereinafter.

3) E-Commerce Microservice System: The dataset is derived from a simulated e-commerce system built upon a microservice architecture by the China Construction Bank Corporation, a top-tier global commercial bank. This system mirrors authentic business traffic and encompasses performance issues extrapolated from real-world scenarios, including various types of performance issues such as CPU, memory, disk, network, and process-related issues. We collected a total of 128 performance issue cases covering 15 types of performance issues. We denote the dataset collected from e-commerce microservice system as $\mathcal{C}$ hereinafter.



(a) Port-level



(b) Service-level

Fig. 6. CDF of the number of port-level and service-level nodes in $\mathcal{A}$.

TABLE I
THE PROPORTION OF CASES WITH INCONSISTENCIES BETWEEN CAUSALITY RELATIONSHIPS AND CALLING RELATIONSHIPS

Dataset	#Inconsistency	#Total	Proportion
$\mathcal{A}$	21	60	35%
$\mathcal{B}$	20	80	25%
$\mathcal{C}$	19	128	15%

Table I presents the quantity and proportion of cases with inconsistent causality and calling relationships mentioned in Section IV-A in datasets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. It is evident that $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are significantly affected by these problems, with $\mathcal{A}$ experiencing a higher prevalence of such problems.

B. Performance Metric and Baselines

Following existing works [2], [12], [16], [30], we select three metrics for evaluation.

1) Performance Metric: Top-k Accuracy (AC@k). AC@k represents the probability that the true root causes of a performance issue case are covered by the top-k recommended root causes. When k is small, the higher the AC@k, the more accurate the method is in identifying the root cause. Due to the small search space, a method with a higher AC@k can greatly improve operators' efficiency in troubleshooting. Given a set of performance issue cases $\mathbb{A}$, AC@k is calculated by (4), where $\mathbb{U}_a$ is the ranked root cause list of case a output by a given method, and $\mathbb{V}_a$ is the correct root cause set of case a. Here we choose k = 1, 3, 5, respectively.

$$AC@k = \frac{1}{|\mathbb{A}|}\sum_{a\in\mathbb{A}}\frac{|\{i \mid \mathbb{U}_a[i] \in \mathbb{V}_a, 0 < i \le k\}|}{min(k, |\mathbb{V}_a|)} \tag{4}$$

Average Top-k Accuracy (Avg@k): Avg@k represents the overall performance of a method in terms of AC@k, calculated by $Avg@k = \frac{1}{k}\sum_{1\le n\le k} AC@n$.

Mean Reciprocal Rank (MRR): MRR focuses on the position of the correct root cause in the list of the ranked root causes output by a method. If the correct answer is not in the ranked list, the rank can be considered positive infinity [2]. The calculation of MRR is shown in (5).

$$MRR = \frac{1}{|\mathbb{A}|}\sum_{a\in\mathbb{A}} 1/\overline{\mathbb{I}_a},\ \mathbb{I}_a = \{i \mid \mathbb{U}_a[i] \in \mathbb{V}_a\} \tag{5}$$

2) Baselines: For a comprehensive evaluation, we compare *MicroDig* with several state-of-the-art baseline approaches. We select six approaches for comparison, namely Microscope [24], ServiceRank [17], MicroHECL [2], MonitorRank [16], TraceRCA [12] and TraceRank [21]. These approaches are all related to causal relationship or calling relationship mining and have achieved superior root cause localization performance in their scenarios. In our experiment, we set each method's parameter best for accuracy. For example, the threshold of the *p-value* is set to 0.02 in Microscope, the hyperparameter α of the PC-algorithm and β of the second-order random walk used in ServiceRank are set to 0.01 and 0.7, respectively. We set the detection window of MicroHECL to 10 minutes after the performance issue occurs, and the correlation threshold is set to 0.5. The weight of the backward edges and the hyperparameter α in MonitorRank are set to 0.2 and 0.85, respectively. We set the main parameters, δ_{fs}, δ_{ad}, and k, in TraceRCA to 0.1, 1, and 100, respectively, to achieve optimal results. Meanwhile, in TraceRank, both d and ρ are configured to 0.5. These methods conduct root cause localization through the performance metrics of each service. Since our dataset only contains the monitoring data of calls, we aggregate each service's call data to obtain its metric data. Specifically, we take the anomaly rate of service as the performance metric of the service, which can be calculated as:

$$\frac{\sum_{p\in S}\left[\sum_{S'\in\mathbb{S}_D, p'\in S'} F_t(p,p') + \sum_{S'\in\mathbb{S}_U, p'\in S'} F_t(p',p)\right]}{\sum_{p\in S}\left[\sum_{S'\in\mathbb{S}_D, p'\in S'} N_t(p,p') + \sum_{S'\in\mathbb{S}_U, p'\in S'} N_t(p',p)\right]}$$

C. Localization Accuracy of MicroDig (RQ1)

To evaluate the effectiveness of *MicroDig*, we conduct a series of experimental studies on the above three datasets. The overall performance is listed in Table II. Experimental results show that *MicroDig* outperforms existing baselines conspicuously. Specifically, on dataset $\mathcal{A}$, *MicroDig* is better than MicroHECL, the best-performed baseline, by 4%, 18.3%, and 23.5% on AC@1, AC@2, and AC@3, respectively. The Avg@3 of *MicroDig* is 81.9%, 16% higher than the second place. In addition, *MicroDig* ranks correct root causes high in the ranking lists, and its MRR reaches 0.78, which is 9.9% higher than other methods at least. On dataset $\mathcal{B}$, *MicroDig* achieves at least a 44.6% improvement

TABLE II
THE ACCURACY OF *MICRODIG* AND THE BASELINE METHODS

Method	Dataset $\mathcal{A}$						Dataset $\mathcal{B}$						Dataset $\mathcal{C}$					
	AC@1	AC@2	AC@3	Avg@3	⇑ Avg@3	MRR	AC@1	AC@2	AC@3	Avg@3	⇑ Avg@3	MRR	AC@1	AC@2	AC@3	Avg@3	⇑ Avg@3	MRR
ServiceRank	50.8%	55.7%	57.4%	54.6%	50.0%	0.55	31.8%	40.9%	56.1%	42.9%	74.1%	0.49	18.4%	40.2%	58.6%	39.1%	88.5%	0.41
MonitorRank	49.2%	61.9%	71.4%	60.8%	34.7%	0.62	34.8%	50.0%	59.1%	48.0%	55.6%	0.51	16.4%	28.9%	41.4%	28.9%	155.0%	0.37
TraceRCA	61.5 %	72.7%	75.8%	70.0%	17.0%	0.70	12.0%	16.9%	22.9%	17.3%	331.8%	0.22	15.6%	25.4%	31.8%	24.3%	203.3%	0.29
TraceRank	16.9%	20.3%	20.3%	19.2%	326.6%	0.20	12.0%	24.2%	40.7%	25.6%	191.8%	0.31	25.0%	30.0%	36.4%	30.5%	141.6%	0.40
Microscope	50.8%	70.4%	75.4%	65.5%	25.0%	0.64	36.4%	57.6%	68.2%	54.1%	38.1%	0.55	13.3%	39.1%	43.0%	31.8%	131.8%	0.40
MicroHECL	61.9%	73.8%	76.2%	70.6%	16.0%	0.71	42.4%	53.0%	74.2%	56.5%	32.2%	0.57	14.1%	30.5%	30.5%	25.0%	194.8%	0.34
MicroDig	**64.4%**	**87.3%**	**94.1%**	**81.9%**	-	**0.78**	**61.3%**	**77.4%**	**85.5%**	**74.7%**	-	**0.74**	**49.2%**	**78.1%**	**93.8%**	**73.7%**	-	**0.70**

TABLE III
THE AVERAGE ELAPSED TIME IN DIAGNOSING A PERFORMANCE ISSUE

Method	Time (s) of dataset $\mathcal{A}$	Time (s) of dataset $\mathcal{B}$	Time (s) of dataset $\mathcal{C}$
MicroScope	3712.93	0.43	14.9
ServiceRank	552.09	0.10	101.2
MicroHECL	81.61	**0.06**	**1.30**
MonitorRank	79.63	0.11	1.51
TraceRCA	420.6	0.09	1.45
TraceRank	821.5	0.19	14.1
MicroDig w/o AD	549.10	0.22	1.81
MicroDig	**24.72**	0.18	1.32

on AC@1 over other baselines. The most commonly used indicators in comprehensive evaluation are Avg@3 and MRR, where *MicroDig* achieves 32.2% and 29.8% improvement over baseline methods at least, respectively. On dataset $\mathcal{C}$, *MicroDig* demonstrates significantly superior performance compared to other baselines, showcasing its stability and robustness in root cause identification.

Since the baseline methods refers to the overall performance indicators of each service, anomalies of a few calls may be submerged in a large number of normal calls, resulting in misdiagnosis. In addition, Microscope's strategy of searching along anomalous paths and retaining only marginal anomalous services is imperfect. When mistakes occur in anomaly detection or the causality mining process, the localization effect can be greatly affected. As for ServiceRank, the causal relationship between services relies entirely on the mining of the PC-algorithm. However, the efficiency and accuracy of the algorithm are low when there are a large number of nodes. The first step in both TraceRCA and TraceRank involves detecting anomalous calls, where the accuracy of call anomaly detection significantly impacts their performance. Over-reliance on call anomaly detection can reduce their adaptability across other Trace datasets. MicroHECL searches the call graph based on the propagation patterns of different types of anomalies. MonitorRank mainly uses calling relationships to construct causal relationships. The localization performances of MicroHECL and MonitorRank are poor due to the inconsistency between calling relationships and causality, as discussed in Section IV.

D. Localization Efficiency of MicroDig (RQ2)

To evaluate the execution efficiency of each method, we give the average elapsed time of each method on datasets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, respectively. As listed in Table III, *MicroDig* achieves the best efficiency on dataset $\mathcal{A}$ with 24.72 seconds per case. The main reason for the efficiency is the construction of the association graph and the applied anomaly detection, which filters out most of the irrelevant service-port items. The low time complexity of the heterogeneous graph construction also contributes a lot. Dataset $\mathcal{B}$ and $\mathcal{C}$ are collected from different systems with fewer microservice instances, resulting in notably shorter average elapsed time for all the approaches. The elapsed time of *MicroDig* is influenced by both the number of microservice instances and the anomalous calls per case. Specifically, the quantity of microservice instances affects the size of the call graph and the computational overhead of random walks. Moreover, the number of anomalous calls impacts the size of the anomalous association graph. For dataset $\mathcal{A}$, collected from Tencent's truly Internet-scale microservice deployments, each case comprises over 8000 real-world microservice instances, resulting in many anomalous calls. Consequently, *MicroDig* and the baseline methods exhibit longer processing times on dataset $\mathcal{A}$ than on datasets $\mathcal{B}$ and $\mathcal{C}$, with *MicroDig* notably faster than baseline methods in comparison. It demonstrates MicroDig's scalability in truly Internet-scale microservice deployments. Through the above analysis, the efficiency of *MicroDig* satisfies the requirement to localize the root cause quickly in online large-scale microservice systems.

E. Ablation Study (RQ3)

In order to evaluate the impact of heterogeneous causal graph, hyperparameter β, and anomaly detection on the accuracy and time consumption of *MicroDig*, we conduct the following ablation experiments.

1) Performance of Heterogeneous Propagation Graph: To show the importance of the heterogeneous propagation graph (HPG) in improving the accuracy of root cause localization, we conduct the following ablation experiments. We replace the HPG from *MicroDig* with a homogeneous propagation graph (with services as nodes and calling relationships as edges). The accuracy of *MicroDig* and *MicroDig* without HPG are listed in Table IV, which shows the importance of the proposed HPG. Specifically, the HPG brings at least 5.6%, 14.4%, 6.8%, 9.1%, and 5.4% improvements on AC@1, AC@2, AC@3, Avg@3, and MRR for *MicroDig* on dataset $\mathcal{A}$, respectively. On dataset $\mathcal{B}$, *MicroDig* achieves a 22.6% improvement on AC@1 over *MicroDig* without HPG. In the most commonly used Avg@3 and MRR, *MicroDig* with HPG is 17.8% and 13.8% higher than *MicroDig* without HPG, respectively. On dataset $\mathcal{C}$, *MicroDig* without the HPG exhibits decreases of 2.3%, 14.8%, 19.6%, 12.2%, and 0.06 in AC@1, AC@2, AC@3, Avg@3, and MRR, respectively, compared to *MicroDig*. The HPG abstracts the

TABLE IV
THE ACCURACY OF *MICRODIG* WHEN THE HETEROGENEOUS PROPAGATION GRAPH (HPG) OR ANOMALY DETECTION (AD) IS REMOVED

Method	Dataset $\mathcal{A}$						Dataset $\mathcal{B}$						Dataset $\mathcal{C}$					
	AC@1	AC@2	AC@3	Avg@3	⇑ Avg@3	MRR	AC@1	AC@2	AC@3	Avg@3	⇑ Avg@3	MRR	AC@1	AC@2	AC@3	Avg@3	⇑ Avg@3	MRR
MicroDig w/o HPG	61.0%	76.3%	88.1%	75.1%	9.1%	0.74	50.0%	66.1%	74.2%	63.4%	17.8%	0.65	46.9%	63.3%	74.2%	61.5%	19.8%	0.64
MicroDig w/o AD	30.5%	61.0%	71.2%	54.2%	51.1%	0.51	46.8%	72.6%	83.9%	67.8%	10.2%	0.66	47.5%	65.0%	75.0%	62.5%	17.9%	0.65
MicroDig	**64.4%**	**87.3%**	**94.1%**	**81.9%**	-	**0.78**	**61.3%**	**77.4%**	**85.5%**	**74.7%**	-	**0.74**	**49.2%**	**78.1%**	**93.8%**	**73.7%**	-	**0.70**

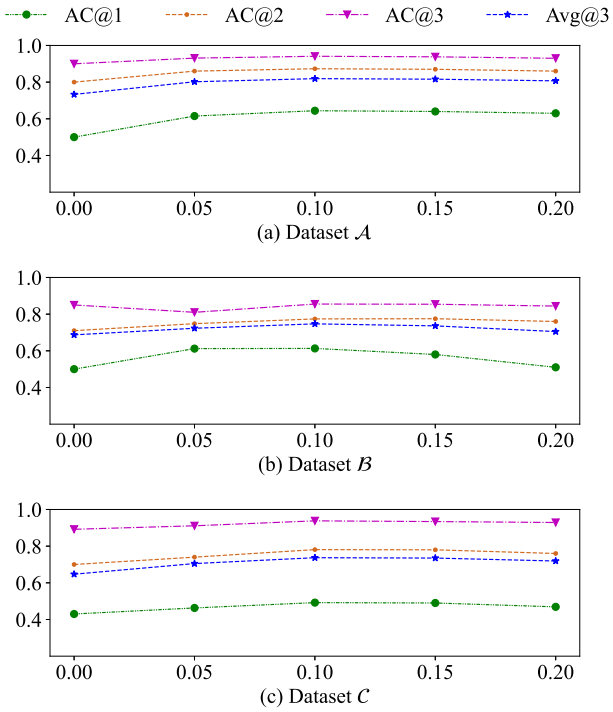


Fig. 7. The accuracy of *MicroDig* as β varies on three datasets.

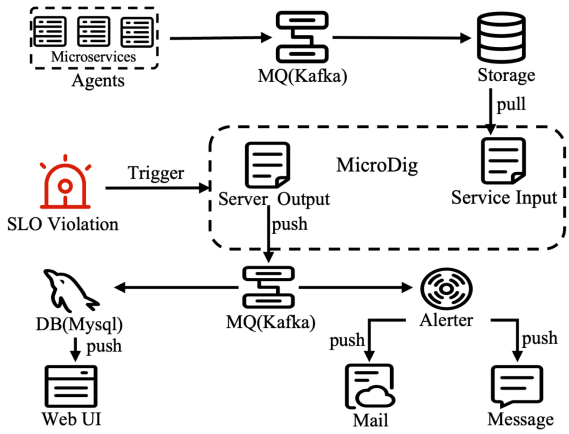


Fig. 8. The deployment environment and running process of *MicroDig*.

call edge as a node, which preserves the data integrity of the call edge, and avoids the inconsistency between causality and calling relationships, achieving more accurate root cause service localization.

2) Impact of Hyperparameter β to HORW: In this part, we will discuss the impact of hyperparameter β in our method. In a complex microservice system, anomaly service detection accuracy cannot be fully guaranteed. At this time, the hyperparameter β is required. It can mitigate the impact of wrong anomaly service detection results on the root cause localization results. From Fig. 7, we can obtain that the best AC@1, AC@2 and AC@3 of *MicroDig* are achieved when β is set to 0.1 on dataset $\mathcal{A}$. For dataset $\mathcal{B}$, the highest level, i.e., AC@1, AC@3 and Avg@3 of *MicroDig* reaches 61.3%, 85.5% and 74.7% when β is set to 0.1, respectively. On dataset $\mathcal{C}$, when β is set to 0.1 and 0.15, *MicroDig*'s AC@1, AC@2, AC@3, and Avg@3 are generally close, although slightly higher when β is set to 0.1.

As demonstrated in Section V-D, the value of β is related to data quality, where higher data quality corresponds to greater anomaly service detection accuracy, and β tends to approach 0, and vice versa. The value of β may vary for different datasets. We set β=0.1 for datasets $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, respectively, because *MicroDig* achieves the best accuracy across all three datasets when β=0.1.

3) Performance of Anomaly Detection: MicroDig detects anomalies before building the heterogeneous propagation graph. Service nodes without anomaly will be removed and will not participate in the subsequent root cause localization steps. As shown in Table IV, anomaly detection evidently affects the results of *MicroDig* on dataset $\mathcal{A}$. *MicroDig* completely surpasses *MicroDig* without anomaly detection (AD) on dataset $\mathcal{A}$. Although the accuracy of these two approaches are comparable on datasets $\mathcal{B}$ and $\mathcal{C}$, the number of service nodes in $\mathcal{B}$ and $\mathcal{C}$ is much smaller than that in $\mathcal{A}$, and the calling relationship is much simpler. As shown in Table III, the operation efficiency of *MicroDig* is much higher than that of *MicroDig* without anomaly detection. The reason for the above results is that anomaly detection helps *MicroDig* avoid the interference of normal service nodes, which greatly reduces the time of building heterogeneous propagation graphs and the HORW.

VII. DISCUSSION

A. Success Story

The approach we proposed has been successfully applied in a system of Tencent's video business. This system includes more than 8000 microservices housing services for tens of millions of users with more than 1 billion requests per day. The deployment environment is shown in Fig. 8.

The agents are responsible for collecting call data in microservices in real time and handing it over to storage. When the monitoring component of the microservice system generates an SLO performance issue, *MicroDig* is triggered. It first requests the calling data from the storage. After analysis and diagnosis, *MicroDig* provides possible root cause services and then sends them to the Message Queue (MQ). Finally, the system sends the

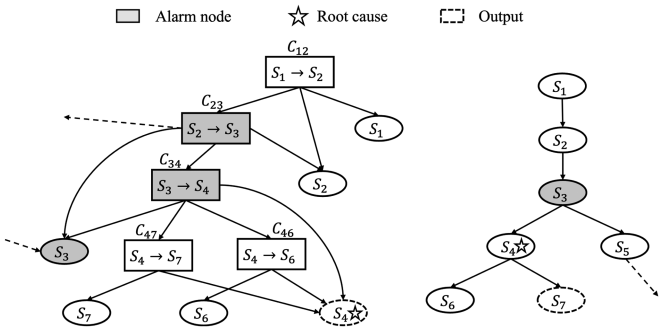


Fig. 9. The root cause localization result of a real case according to a heterogeneous propagation graph (left) or a call graph (right).

performance issue and candidate root cause list to operators in the form of web pages, emails, and messages.

Before deploying our approach, root cause localization is labor-intensive and error-prone. Usually, it takes operators 30-60 minutes to manually localize the root cause. Therefore, performance issues cannot be mitigated in a timely and accurate manner, which may cause user complaints. After applying *MicroDig*, operators are freed from the complicated troubleshooting work. They can accurately diagnose the root cause and greatly shorten the root cause localization time to less than one minute.

Take a real-world performance issue that occurred at 20:42 on December 22, 2021, as an example. The sudden traffic increase on S_3 leads to a service performance issue. Since there are a large number of service nodes related to this performance issue, Fig. 9 only shows part of the nodes. The star symbol and the gray mark in this figure represent the real root cause and issue service, respectively. In addition, the dashed ellipse represents the root cause output by the root cause localization method. It can be seen from the right picture of Fig. 9 that the calling relationship-based methods are more inclined to approach downstream nodes to find the root cause according to the transition probability. This is why it skips the real root cause S_4 and walks toward S_7 repeatedly. The heterogeneous graph constructed by *MicroDig* is shown in the left picture of Fig. 9. Because each call is abstracted as a call node, three issue nodes are related to S_3 in the heterogeneous propagation graph. *MicroDig* takes the issue call node (i.e., C_{23}, C_{34}) as the entry node of root cause localization. With the call nodes, neither (S_4, S_7) nor (S_4, S_6) is upstream/downstream causal relationships. Although there are some false positives in anomaly detection, *MicroDig* finds the correct root cause by evaluating the similarity of the calling nodes during the random walk. This case is associated with a total of 5740 services. *MicroDig* pinpointed the root cause of the performance issue in 58 seconds, proving that our solution is effective and efficient.

B. Lessons Learned

After applying *MicroDig* in Tencent, we notice a real case due to parameter transmission between two downstream calls sent by the same service. Specifically, denote the upstream service as A and it calls two services in turn, namely B and C. In this case, A transmitted the value returned by $A \to^{call} B$ as a parameter for $A \to^{call} C$. Unfortunately, although $A \to^{call} B$ did not throw exceptions, the return result of $A \to^{call} B$ was invalid or broken, which caused an anomaly of $A \to^{call} C$. The effect of such an anomaly is not reflected in the calling relationship. As a result, *MicroDig* only localized the performance issue service C, because C was the service that produced an anomaly directly. In the follow-up troubleshooting, operators continued to check the error log supplemented by prior knowledge and found that the root cause was an error in the incoming parameters. Although this situation is rare, in the future, we will gather more information and knowledge to further enhance *MicroDig*.

Theoretically, integrating more data sources, like metrics and logs, could enhance *MicroDig*'s accuracy. For now, *MicroDig* can incorporate metrics, in addition to traces, as crucial inputs for constructing its anomalous association graph. We will expand the model to integrate more data sources, such as logs, to further improve *MicroDig*'s accuracy in future work.

C. Threats to Validity

The major threat to interval validity lies in our implementation of the four baseline methods, which are based on our understanding as they are not publicly available. To mitigate this threat, several authors have carefully checked the code.

The threat to external validity mainly lies in the two adopted datasets, which may not represent various microservice systems. In the future, we will apply *MicroDig* to more different companies to mitigate this threat. On the other hand, as we focus on the root cause localization at the service level, detailed data, such as metrics of physical machines, are omitted. We leave the in-depth root cause localization of fusing multiple types of data as future work.

The threat to construction validity mainly lies in the used hyper-parameters and metrics. To reduce this threat, on the one hand, we select the hyper-parameters of *MicroDig* and the baseline methods through grid search. On the other hand, we apply the most widely used effectiveness and efficiency metrics following the existing works [2], [23], [30], [31].

VIII. Conclusion

In this paper, we propose an approach named *MicroDig* for automatic root cause localization of performance issues in large-scale microservice systems. *MicroDig* identifies the association calls and builds a heterogeneous propagation graph based on the dynamic microservice invocations. It then applies HORW to find the patterns of anomaly propagation and pinpoint the culprit service. The effectiveness and efficiency of *MicroDig* are proved using both real-world performance issues collected from Tencent and manually injected ones collected from Train-Ticket and China Construction Bank Corporation. *MicroDig* achieves superior performance compared to four popular root cause localization methods. Moreover, we have shared our success stories and learned lessons from *MicroDig*'s deployment in Tencent. Specifically, *MicroDig* significantly shortens the performance issue diagnosis time from 30-60 minutes to less than one minute. In future work, metrics and logs of microservices can be fused to pinpoint the root cause more accurately.

REFERENCES

[1] J. Lewis and M. Fowler, "Microservices," 2014, [Online]. Available: https://martinfowler.com/articles/microservices.html

[2] D. Liu et al., "MicroHECL: High-efficient root cause localization in large-scale microservice systems," in *Proc. 43rd IEEE/ACM Int. Conf. Softw. Eng.*, Madrid, Spain, May 25–28, 2021, pp. 338–347.

[3] P. Wang et al., "CloudRanger: Root cause identification for cloud native systems," in *Proc. 18th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, Washington, DC, USA, May 1–4, 2018, pp. 492–502.

[4] L. Weng, Y. Hu, P. Huang, J. Nieh, and J. Yang, "Effective performance issue diagnosis with value-assisted cost profiling," in *Proc. 18th Eur. Conf. Comput. Syst.*, G. A. D. Luna, L. Querzoni, A. Fedorova, and D. Narayanan, Eds, Rome, Italy, May 8–12, 2023, pp. 1–17.

[5] J. Chen et al., "An empirical investigation of incident triage for online service systems," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.*, Montreal, QC, Canada, May 25–31, 2019, pp. 111–120.

[6] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA, USA: O'Reilly Media, Inc, 2016.

[7] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "MicroRCA: Root cause localization of performance issues in microservices," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, Budapest, Hungary, Apr. 20–24, 2020, pp. 1–9.

[8] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Causal inference techniques for microservice performance diagnosis: Evaluation and guiding recommendations," in *Proc. IEEE Int. Conf. Autonomic Comput. Self-Organizing Syst.*, Washington, DC, USA, Sep. 27 - Oct. 1, 2021, pp. 21–30.

[9] R. Wang and S. Ying, "SaaS software performance issue diagnosis using independent component analysis and restricted boltzmann machine," *Concurr. Comput. Pract. Exp.*, vol. 32, no. 14, 2020, Art. no. e5729.

[10] Y. Han, "Enterprise operation's top 3 factors of lengthy MTTR and ways to reduce them," 2019, [Online]. Available: https://community.ibm.com/community/user/aiops/blogs/yok-han1/2019/06/12/enterprise-operations-top-3-factors-of-lengthy-mtt

[11] P. Liu et al., "Unsupervised detection of microservice trace anomalies through service-level deep Bayesian networks," in *Proc. 31st IEEE Int. Symp. Softw. Rel. Eng.*, Coimbra, Portugal, Oct. 12–15, 2020, pp. 48–58.

[12] Z. Li et al., "Practical root cause localization for microservice systems via trace analysis," in *Proc. 29th IEEE/ACM Int. Symp. Qual. Service*, Tokyo, Japan, Jun. 25–28, 2021, pp. 1–10.

[13] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: Practical and scalable ML-driven performance debugging in microservices," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, USA, Apr. 19–23, 2021, pp. 135–151.

[14] X. Guo et al., "Graph-based trace analysis for microservice architecture understanding and problem diagnosis," in *Proc. 28th ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, USA, Nov. 8–13, 2020, pp. 1387–1397.

[15] Á. Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, and V. Muntés-Mulero, "Graph-based root cause analysis for service-oriented and microservice architectures," *J. Syst. Softw.*, vol. 159, 2020, Art. no. 110432.

[16] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," in *Proc. ACM SIGMETRICS/Int. Conf. Meas. Model. Comput. Syst.*, Pittsburgh, PA, USA, Jun. 17–21, 2013, pp. 93–104.

[17] M. Ma, W. Lin, D. Pan, and P. Wang, "ServiceRank: Root cause identification of anomaly in large-scale microservice architectures," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 5, pp. 3087–3100, Sep./Oct. 2022.

[18] P. Chen, Y. Qi, P. Zheng, and D. Hou, "CauseInfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems," in *Proc. IEEE Conf. Comput. Commun.*, Toronto, Canada, Apr. 27 - May 2, 2014, pp. 1887–1895.

[19] X. Zhou et al., "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, pp. 243–260, Feb. 2021.

[20] MicroDig, 2022, [Online]. Available: https://github.com/hburning/MicroDig

[21] G. Yu, Z. Huang, and P. Chen, "TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems," *J. Softw., Evol. Process*, vol. 35, 2021, Art. no. e2413.

[22] H. Nguyen, Z. Shen, Y. Tan, and X. Gu, "FChain: Toward black-box online fault localization for cloud systems," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, Philadelphia, Pennsylvania, USA, Jul. 8–11, 2013, pp. 21–30.

[23] M. Ma, W. Lin, D. Pan, and P. Wang, "MS-Rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications," in *Proc. IEEE Int. Conf. Web Serv.*, Milan, Italy, Jul. 8–13, 2019, pp. 60–67.

[24] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *Proc. 16th Int. Conf. Service-Oriented Comput.*, Hangzhou, China, Springer, Nov. 12–15, 2018, pp. 3–20.

[25] Jaeger, 2022, [Online]. Available: https://www.jaegertracing.io/

[26] Zipkin, 2022, [Online]. Available: https://zipkin.io/

[27] OpenTelemetry, 2022, [Online]. Available: https://opentelemetry.io/

[28] J. Pearl, *Causality: Models, Reasoning, and Inference*, 2nd ed., Cambridge, U.K.: Cambridge Univ. Press, 2009.

[29] Kubernetes, 2022, [Online]. Available: https://kubernetes.io/

[30] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "Automap: Diagnose your microservice-based web applications automatically," in *Proc. Web Conf.*, Taipei, Taiwan, Apr. 20–24, 2020, pp. 246–258.

[31] Y. Meng et al., "Localizing failure root causes in a microservice through causality inference," in *Proc. 28th IEEE/ACM Int. Symp. Qual. Service*, Hangzhou, China, Jun. 15–17, 2020, pp. 1–10.

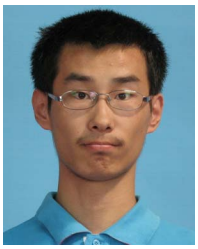
Lei Tao received the MS degree in software engineering from Nankai University, Tianjin, China, in 2022. He is currently working toward the PhD degree with the College of Software at Nankai University, Tianjin, China. His research interests include anomaly detection and failure diagnosis.

Xianglin Lu received the BS degree in information security from the School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China, in 2020. She is currently working toward the MS degree with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China. Her research interests include anomaly detection and failure diagnosis.

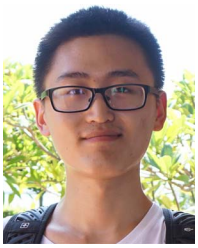
Shenglin Zhang (Member, IEEE) received the BS degree in network engineering from the School of Computer Science and Technology, Xidian University, Xi'an, China, in 2012 and the PhD degree in computer science from Tsinghua University, Beijing, China, in 2017. He is currently an associate professor with the College of Software, Nankai University, Tianjin, China. His current research interests include failure detection, diagnosis, and prediction for service management.

Jiaqi Luan is currently working toward the senior undergraduate degree with the College of Software, Nankai University, Tianjin, China. Her research interests include anomaly detection, root cause localization, and data security.

Yingke Li received the BS degree in software engineering from the School of Information Engineering, Minzu University of China, Beijing, China, in 2018. She is currently working toward the MS degree with the College of Software, Nankai University, Tianjin, China. Her research interests include anomaly detection and failure diagnosis.



Mingjie Li received the BS degree in computer science from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2018. He is currently working toward the PhD degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include root cause localization and software engineering.



Zeyan Li received the BS degree from Tsinghua University, Beijing, China, in 2018. He is currently working toward the PhD degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His primary research focuses on artificial intelligence for IT operations.

Qingyang Yu received the BS degree in software engineering from Shandong University, Jinan, China, in 2014 and the MS degree in computer technology from the University of Chinese Academy of Sciences, Beijing, China, in 2017. He is currently working toward the PhD degree with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests lie in anomaly detection and failure diagnosis.

Hucheng Xie received the BS and MS degrees in computer science from Harbin Institute of Technology, Harbin, China, in 2013 and 2015, respectively. His research interests include DevOps observation and telemetry. He is currently an engineer with Tencent, Inc., Shenzhen, China.

Ruijie Xu received the BS degree in computer science from Lingnan Normal University, Zhanjiang, China, in 2016. His research interests include AIOps observation and telemetry. He is currently an algorithm researcher with Tencent, Inc., Shenzhen, China.

Chenyuan Hu received the BS degree in computer science from the East China University of Technology, Nanchang, China, in 2012. His research interests include cloud-native observability and service management in general. He is currently a senior engineer with Tencent, Inc., Shenzhen, China.

Canqun Yang received the MS and PhD degrees from the National University of Defense Technology in 1995 and 2008, respectively. He is currently a researcher with the College of Computer Science, National University of Defense Technology, and serves as the director of the National Supercomputing Center in Tianjin. His primary research areas include high-performance computing and industrial software.

Dan Pei (Senior Member, IEEE) received the BE and MS degrees in computer science from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 1997 and 2000, respectively, and the PhD degree in computer science from the Computer Science Department, University of California, Los Angeles (UCLA) in 2005. He is currently an Associate Professor with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include network and service management in general. He is an ACM senior member.