Graph-Based Root Cause Localization in Microservice Systems with Protection Mechanisms

Haitao Zhang[1], Wei Tian[1], Neng Yang[1], and Yepeng Zhang[1]

[1]Beijing University of Posts and Telecommunications School of Computer Science

December 16, 2022

Abstract

Nowadays, the protection mechanisms are introduced into microservice systems to ensure the stable operation of services. However, existing approaches ignore the impact of protection mechanisms on the root cause localization of abnormal services. Specifically, the circuit breaking and rate limiting mechanisms can refuse service requests and thus change the way of anomaly propagation. Moreover, different service request frequencies and response time make service dependencies change dynamically, resulting in different probabilities of anomaly propagation among services. In this paper, we propose a novel framework named MicroGBPM to locate the root cause of abnormal services, which considers the impact of the protection mechanisms. We model anomaly propagation among services as a dynamically constructed service attributed graph with metrics and traces when a failure occurs. To eliminate the impact of the protection mechanisms, we design a two-stage dynamic calibration strategy to adjust the probability of anomaly propagation among services. Then we propose a random walking approach to calculate the root cause results by using the PageRank algorithm. The experimental results show that MicroGBPM improves the accuracy of root cause localization compared to other approaches in microservice systems with protection mechanisms.

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

SPECIAL ISSUE-TECHNOLOGICAL

Graph-Based Root Cause Localization in Microservice Systems with Protection Mechanisms

Wei Tian | Haitao Zhang | Neng Yang | Yepeng Zhang

School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China

Correspondence
Haitao Zhang, School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China
Email: zht@bupt.edu.cn

Funding Information
The National Key Research and Development Plan under Grant No. 2020YFF0305501 and the National Natural Science Foundation of China under Grant No. 62172050

Abstract

Nowadays, the protection mechanisms are introduced into microservice systems to ensure the stable operation of services. However, existing approaches ignore the impact of protection mechanisms on the root cause localization of abnormal services. Specifically, the circuit breaking and rate limiting mechanisms can refuse service requests and thus change the way of anomaly propagation. Moreover, different service request frequencies and response time make service dependencies change dynamically, resulting in different probabilities of anomaly propagation among services. In this paper, we propose a novel framework named MicroGBPM to locate the root cause of abnormal services, which considers the impact of the protection mechanisms. We model anomaly propagation among services as a dynamically constructed service attributed graph with metrics and traces when a failure occurs. To eliminate the impact of the protection mechanisms, we design a two-stage dynamic calibration strategy to adjust the probability of anomaly propagation among services. Then we propose a random walking approach to calculate the root cause results by using the PageRank algorithm. The experimental results show that MicroGBPM improves the accuracy of root cause localization compared to other approaches in microservice systems with protection mechanisms.

KEYWORDS:
microservice, root cause localization, attributed graph, protection mechanism, random walking

1 | INTRODUCTION

A growing number of applications are embracing the microservice architecture [1,2]. The microservice architecture uses a modular approach to spilt large software applications into hundreds or thousands of smaller, more independent, and more manageable microservices [3,4]. Meanwhile, there are complex dependencies between microservices. Multiple microservices work together to respond to user requests. Once a single service goes down, it is likely to trigger a cascading failure that can cause the entire system to crash. When multiple services are abnormal at the same time, site reliability engineers need to accurately locate the root cause of abnormal services to avoid huge economic losses [5,6]. Therefore, it is meaningful and challenging to accurately locate the root cause of abnormal services in large microservice environment.

Microservice architectures usually introduce protection mechanisms to enhance the stability of services, such as the circuit breaking and rate limiting mechanisms. When the number of abnormal requests from the upstream service to the downstream service reaches a certain threshold, the upstream service will perform the circuit breaker to avoid causing service cascading failure. When the number of service requests exceeds the threshold, the service turns on the rate limiting mechanism. Excess service requests are rejected to avoid service crashes caused by high concurrent requests. Although these protection mechanisms reduce the probability of service failure, they also affect the way of anomaly propagation. The

impact of different levels of protection mechanisms also varies. In addition, service dependencies are related to service access frequency and response time. Service access frequency is different. Service response time also varies between different types of services. These differences make service dependencies change dynamically, resulting in different probabilities of anomaly propagation among services. The protection mechanisms and the dynamicity of service dependencies further increase the complexity of root cause localization. There is still a problem that it is difficult to locate the root cause of abnormal services accurately in microservice systems with protection mechanisms.

In recent years, many approaches on the root cause localization of abnormal services have been proposed, including trace analysis and service dependency graph. On the one hand, the trace analysis approaches use service metrics and traces to locate the root causes by calculating the degree of service anomalies, e.g., MEPFL [7], Diagnose [8], T-Rank [9]. However, these approaches suffer from some practical issues. Specifically, Diagnose and T-Rank ignore the dependencies between services and do not take into account the propagation of anomalies. MEPFL locates the root causes by training a supervised learning model, but it is impractical to guarantee accuracy in different microservice environments. On the other hand, the service dependency graph approaches use causal relationships [10–12] or service calls [13–15] to construct service dependency graphs. These approaches use metrics, logs and traces to calculate service anomaly scores and locate the root cause of abnormal services by traversing the service dependency graph. However, none of the above approaches take into account the impact of protection mechanisms. When the protection mechanisms are open, the collected traces may be inaccurate. It is difficult to accurately locate the root causes by trace analysis approaches. In addition, the service correlation will be reduced, making it difficult to accurately locate the root causes when using the service dependency graph approaches. In summary, the above approaches are difficult to accurately locate the root cause of abnormal services in microservice systems with protection mechanisms.

In this paper, we propose a novel framework named MicroGBPM to locate the root causes of abnormal services. The main idea of MicroGBPM is to utilize metrics and traces and eliminate the effect of the protection mechanisms to locate the root causes. Firstly, we model the process of anomaly propagation as a dynamically constructed service attributed graph. We use metrics and traces to calculate service anomaly scores and the probability of anomaly propagation among services when an anomaly is detected. Secondly, we design a two-stage dynamic calibration strategy to eliminate the effect of the protection mechanisms on anomaly propagation. When the circuit breaking mechanism is open, we use the in-neighbor service metrics to adjust the current service metrics. We calculate the probability of rejected requests to quantify the impact of the rate limiting mechanism on service correlation. Finally, we propose a random walking approach based on the PageRank algorithm and design a series of flexible transitions to make the results of root cause localization more accurate. We evaluate the accuracy of MicroGBPM based on two open-source microservices systems Train-Ticket[†] and Bookinfo[‡]. We inject different types of faults separately and analyze the accuracy under one or two root causes. The experimental results show that MicroGBPM can improve the accuracy of root cause localization compared to other state-of-the-art approaches in microservice systems with protection mechanisms. The results also confirm the effectiveness of the protection mechanism calibration strategy, which can significantly improve the accuracy of root cause localization results.

In summary, our contributions are threefold:

- As far as we know, we are the first to solve the root cause localization problem in the microservice systems with protection mechanisms, in which our objective is to eliminate the effects of protection mechanisms and improve the accuracy of root cause localization.
- We propose a new framework to locate the root cause of abnormal services effectively by using metrics, traces and service attributed graph in microservice systems with protection mechanisms.
- We evaluate MicroGBPM through different types of failures and different levels of protection mechanisms. The experimental results show that MicroGBPM improves the accuracy of root cause localization compared to other approaches in microservice systems with protection mechanisms.

The rest of this paper is organized as follows. We present the background knowledge related to MicroGBPM in Section 2. We clarify the motivation and define the problem in Section 3. In Section 4, we introduce the system model and the approach of root cause localization in detail. We describe the experiment and analyze the results in Section 5. We summarize the related work in Section 6. We conclude the work of this paper in Section 7.

[†] https://github.com/FudanSELab/train-ticket
[‡] https://istio.io/latest/docs/examples/bookinfo/

2 | BACKGROUND

2.1 | Microservice observability

In IT and cloud computing, observability is the ability to measure a system's current state based on the data it generates, such as logs, metrics, and traces. Every node (physical or virtual machine), microservice, and container generates records of each activity. The goal of observability is to understand what is happening in all of these environments and among the technologies so that system failures can be detected and resolved to keep the system efficient and reliable. In this paper, we use service metrics and traces for root cause localization. We collect the metrics to determine the current abnormal state of the service. Meanwhile, a back-end request may have to go through multiple services before the final response reaches the client. We can better analyze system anomalies and solve the root cause locating problems if we can track each request and analyze information such as response time, status codes, etc.

In this paper, we use Prometheus[§] to collect service metrics, including CPU usage, memory usage, network receive/send throughput. We use the response time collected from traces as one of the metrics as well. We use these metrics to determine the current level of service anomalies. Meanwhile, all invocations realizing the same user request form a trace. A trace consists of multiple spans. Each span includes information such as instance, operation, traceId, parentId, HTTP status code and so on. We collect traces using OpenTelemetry[¶], which is a set of APIs, SDKs, tools, and integrations designed to create and manage telemetry data. From the trace data, we can get the response time, access frequency, and service call graph. We can locate root causes at service instance level based on the tracing information. Figure 1 shows a series of spans and their relationships within a trace in the Bookinfo system.

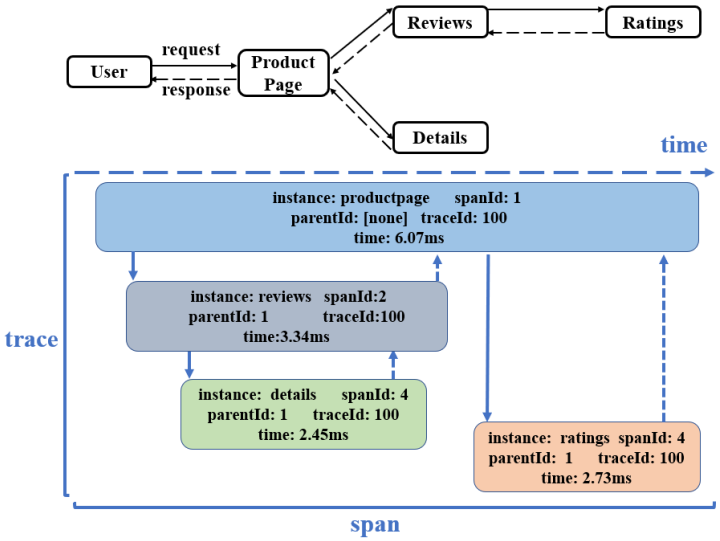

FIGURE 1 An example of microservice trace in the Bookinfo system

2.2 | Microservice protection mechanisms

In real production environments, systems often face high concurrent situation. Burst requests can lead to resources such as network bandwidth, CPU, and memory not being able to meet the needs of business requests. In addition, failures can propagate further causing greater impact due to the complex dependencies between services. Therefore, site reliability engineers introduce the protection mechanisms in order to ensure services stable operation. We introduce two common protection mechanisms in this section, including circuit breaking and rate limiting mechanisms.

Circuit Breaking Mechanism: When the number of abnormal requests from the upstream service to the downstream service reaches a certain threshold, the upstream service will open the circuit breaker to avoid causing service cascading failure. The circuit breaker includes the following three states, as shown in the Figure 2.

[§] https://prometheus.io/
[¶] https://opentelemetry.io/

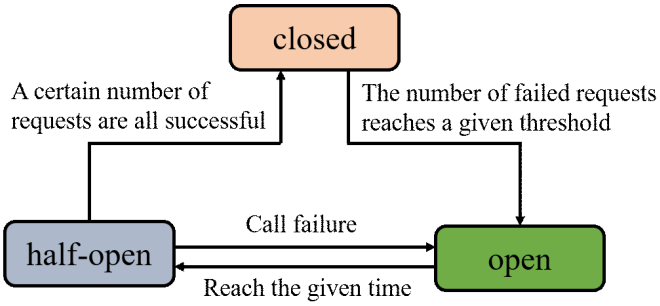

FIGURE 2 Circuit breaker status and transformation conditions

- Closed: The circuit breaker is in the closed state. When the number of abnormal requests accumulates and reaches the threshold, the circuit breaker is activated.
- Open: The circuit breaker is in the open state. At this point, the service requests from upstream to downstream will return errors directly. After KT seconds, the circuit breaker enters the half-open state.
- Half-open: When the circuit breaker is in the half-open state, the upstream service allows some service requests to be forwarded to the downstream service. If all of them are successful, the downstream service is considered to be restored. At this point, the circuit breaker will be closed. Otherwise, the downstream service is considered to have not recovered and the circuit breaker will return to the open state.

Rate Limiting Mechanism: The rate limiting mechanism is designed to protect service from excessive transient traffic that can cause service crashes and unavailability. The common rate limiting algorithms include the counter algorithm, the leaky bucket algorithm, the token bucket algorithm, and the sliding window algorithm. We use the leaky bucket algorithm as an example to analyze the impacts of the rate limiting mechanism on root cause localization in MicroGBPM. The leaky bucket algorithm can be roughly described as water injection and leakage process. The water flows out of the bucket at a certain rate. When the water exceeds the capacity of the bucket, the excess water will be discarded. Similarly, its main purpose in the microservice systems is to control the service access rate and smooth out the burst traffic. Note that we use the leaky bucket algorithm as an example in MicroGBPM. Similarly, we can eliminate the effects of other rate limiting algorithms when they are introduced into the microservice systems.

2.3 | PageRank algorithm

The PageRank algorithm is proposed to calculate the importance of Internet pages. The basic idea of the PageRank algorithm is to define a random walking model based on a directed graph. It describes the behavior of a walker visiting each node randomly along the directed graph. The probability of visiting each node in the limit case converges to a smooth distribution, indicating the importance of the node. The PageRank algorithm [16] is described as follows:

$$X = d * \mathbf{P} * X + (1 - d) * U, \tag{1}$$

where X presents the score vector, $\mathbf{P}$ denotes transition matrix, U denotes the additional teleportation vector, and d denotes the damping parameter. Similarly, we can simulate service anomaly propagation among services by using the PageRank algorithm. The importance of the nodes represents the probability that the services are the root causes.

3 | MOTIVATION AND PROBLEM DESCRIPTION

In this section, we describe the challenges of the root cause localization of abnormal services and analyze the impact of protection mechanisms by case in detail. Then, we formulate the root cause localization of abnormal services.

3.1 | Motivation

There are complex dependencies between services. The anomaly of a single service may cause a cascading failure. It is not reasonable to select the node with the largest degree of abnormality as the root cause of abnormal services. As shown in Figure 3, the anomaly of ms_4 and ms_5 causes the

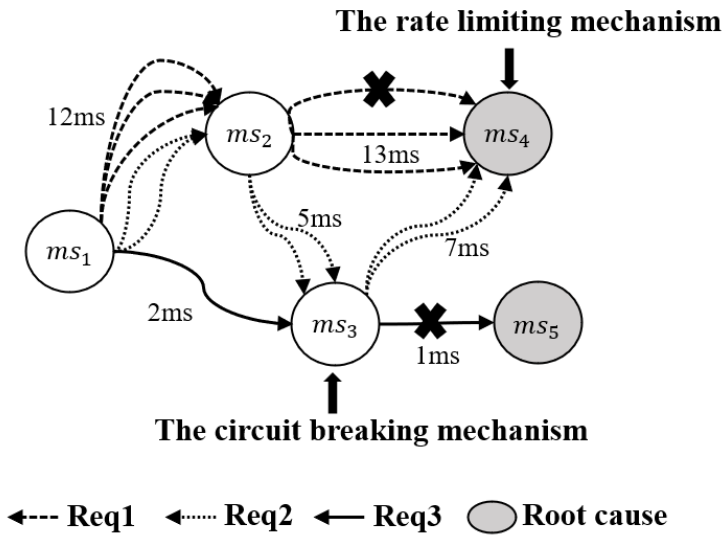

FIGURE 3 An example of a microservice system that contains the protection mechanisms, different service access frequency, and different service response time

simultaneous anomaly of multiple services. However, the anomaly score of ms_2 may be greater than the anomaly score of ms_4 or ms_5. Therefore, the service with the larger anomaly score is not necessarily the root cause, and we need to further consider the service dependencies.

The service dependency graphs are dynamically changing. The request frequency and response time among services are different. These factors lead to the differences in service correlation and anomaly propagation probability. As shown in Figure 3, $Req1$ is requested 3 times with a response time of 25ms. $Req2$ is requested 2 times with a response time of $24ms$. $Req3$ is requested 1 time with a response time of $3ms$. Different service dependency levels can lead to different root cause location results when using the service dependency graph approaches. Therefore, we need to dynamically construct service dependency graphs to locate the root causes more accurately.

When the circuit breaking mechanism is activated, the service requests will be rejected between services. When the rate limiting mechanism is open, excess service requests will be rejected between services. The correlation between services is reduced, making it difficult to locate the root causes. We take Figure 3 as an example. When ms_3 starts the circuit breaking mechanism, the service dependency between ms_3 and ms_5 decreases. However, due to the presence of ms_4, ms_3 still behaves abnormally. Considering ms_4 and ms_5 as the root causes, the probability of ms_5 being the root cause obtained decreases when the service dependency between ms_3 and ms_5 decreases. At this point, it is difficult to get ms_5 as the root cause, making the root cause location result inaccurate. In addition, the correlation between ms_2 and ms_4 is reduced due to the rate limiting mechanism, thus affecting the accuracy of root cause localization. Therefore, we need to eliminate the impact of protection mechanisms on root cause localization.

3.2 | Problem description

In this paper, we work on locating the root cause of abnormal services more accurately in microservice systems with protection mechanisms. We can collect the m types of metrics of service i and N traces, defined as $\mathbf{M_i} = \{M_{i1}, M_{i2}, \cdots, M_{im}\}$ and $T = \{T_1, T_2, \cdots, T_N\}$. If there exists an anomaly, we use the current service as the starting point to determine whether the upstream and downstream services are abnormal. We model the anomaly propagation among services as a service attributed graph $G(V, E)$, where V denotes the set of services and E denotes the set of edges. Each e_{ij} is set to 1 if there exists service call between service i and service j. For each service graph node v_i, it is described by a vector that denotes the anomaly score, i.e., $v_i = (\delta_{i1}, \delta_{i2})$, where δ_{i1} denotes the metric anomaly score of service i and δ_{i2} denotes the trace anomaly score of service i. If $e_{ij} = 1$, we calculate the weight W_{ij} by service metric correlation. In addition, we eliminate the effect of the protection mechanisms on the service attributed graph weights. Therefore, our goal is to rank the abnormal services by traversing the attributed graph $G(V, E)$ and select the abnormal service with the highest score.

4 | SYSTEM DESIGN AND APPROACH

In this section, we describe the components of MicroGBPM in detail, as shown in the Figure 4. MicroGBPM contains 5 modules, namely anomaly detection module, data preparation module, attributed graph construction module, protection mechanism calibration module, and abnormal services ranking module. Once the anomaly detection module (Section 4.1) detects the anomaly, the data preparation module (Section 4.2) processes the collected data. After that, MicroGBPM uses metrics and traces to calculate service anomaly scores and service correlation, and dynamically construct service attributed graphs (Section 4.3). The protection mechanism calibration module (Section 4.4) uses traces to determine whether the circuit breaking and rate limit mechanisms are open and calibrates the weight of edges in the service attributed graph. Finally, MicroGBPM calculates the root cause ranking results by using the PageRank algorithm in the abnormal services ranking module (Section 4.5). Table 1 lists the primary notations in this section.

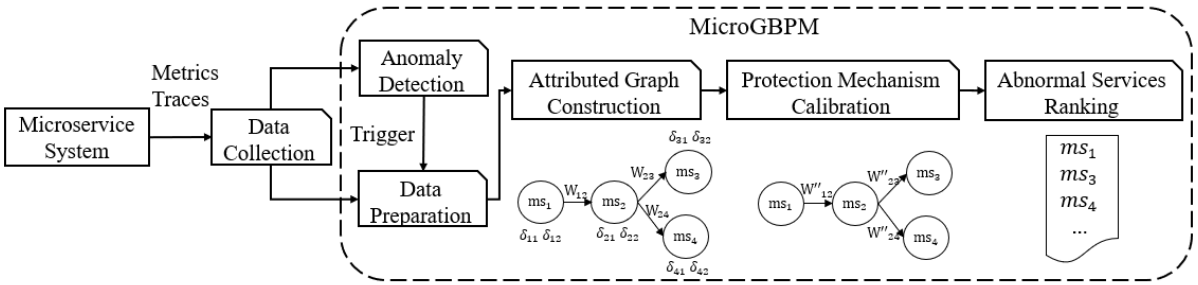

FIGURE 4 Overview of MicroGBPM components and root cause localization workflow

4.1 | Anomaly detection

Before locating the root causes of the abnormal services, we have to determine whether there is an anomaly in the current system. The response time is the intuitive representation of service status. Considering the existence of periodic fluctuations, it is difficult for traditional statistical methods such as the 3-sigma principle to accurately determine whether the response time is abnormal. In addition, considering the inconsistent response time of different service requests, it is difficult to find a uniform standard to determine whether the service response time is abnormal. For example, the response time of $Req1$ and $Req3$ in Figure 3 are inconsistent.

In this paper, we use a k-means model to cluster traces of the same service. Each trace passes through multiple service instances. We encode each trace for easy clustering. If the trace goes through the current service, it is defined as 1. Otherwise, it is defined as 0. For example, we encode the $Req1$ in Figure 3 as $[1, 1, 0, 1, 0]$. We calculate the Euclidean distance between encoded vectors. We automatically infer the optimal number of clusters by the gap statistic method [17]. After that we determine whether the response time of each trace is abnormal by using the KDE (Kernel Density Estimation) algorithm [18,19], which is very suitable for anomaly detection. When persistent anomalies of service response time are detected, the data preparation module is triggered.

4.2 | Data preparation

The anomaly detection module determines whether an anomaly has occurred but cannot locate the root causes. Once the root cause localization phase is triggered, the data preparation module will extract the information from the traces. The data preparation module divides the traces into normal traces and abnormal traces by the response time. It also records the service instances of each trace and counts the number of each kind of trace. The data preparation module provides the processed information to the attributed graph construction module.

4.3 | Attributed graph construction

In this section, we dynamically construct service attributed graphs to represent the propagation of anomalies during a detection cycle. We calculate the metric and trace anomaly scores of each abnormal service. When there exist requests between services, we calculate the weight of the directed edges by using the correlation of the metrics. The implementation is described as follows.

TABLE 1 Notations

Notation	Definitions
N, n	the number of traces, the number of services
m, c	the type number of metrics, the collected number of each type metric
M_{ik}	the k-th metric set of service i
ES_{ik}	the k-th metric anomaly severity set of service i
δ_{i1},δ_{i2}	the metric anomaly score and trace anomaly score of service i
e_f^i, e_p^i, n_f^i	the critical statistics of SBFL technique
$corr_{ijk}$	the k-th correlation score from service i to j
W_{ijk}	the k-th calibration weight from service i to j
k_1, k_2	the circuit breaker opening and closing thresholds
KT	the duration opening time of the circuit breaker
$Tsize_{ii}$	the number of requests between service i and j
$Hsize_{ij}$	the number of rejected requests due to rate limiting mechanism between service i and j
θ_{ij}	the rejected probability due to rate limiting mechanism between service i and j
RM_{ij}	the metrics correlation correction factor of service i to j
ρ_1,ρ_2	the fallback coefficient, the self coefficient
$\mathbf{W}, \mathbf{P}$	the correlation matrix, the transition matrix
IS, S	the suspicious score vector, the anomaly score vector
I, O	the set of in-neighbor and out-neighbor services
U	the additional teleportation vector
X	the result of root cause ranking vector

Metric Anomaly Score: We use the mean μ_{ik} and standard deviation σ_{ik} of the service metrics to calculate service anomaly severity [20–24]. The μ_{ik} is the expected normal value and the σ_{ik} indicates that the metric deviates from the mean. We define the k-th metric set of service i as $M_{ik} = \{M_{ik}^1, ..., M_{ik}^c\}$ and the k-th metric anomaly severity set of service i as $ES_{ik} = \{ES_{ik}^1, ..., ES_{ik}^c\}$. The service anomaly severity ES_{ik}^t of metric M_{ik}^t is defined as:

$$ES_{ik}^t = \frac{|M_{ik}^t - \mu_{ik}|}{\sigma_{ik}}. \tag{2}$$

To eliminate the impact of the metrics fluctuation, we calculate the mean and standard deviation of the normal metrics in the previous period and the same time of the previous day respectively. The μ_{ik} and σ_{ik} are dynamically updated to ensure the accuracy of the calculation. We calculate the average anomaly scores of k-th metric over a detection period and choose the largest one as the anomaly score of service i. The metric anomaly score is defined as:

$$\delta_{i1} = \max \frac{\sum_{t=1}^{c} ES_{ik}^t}{c},\ k = 1, 2, ..., m. \tag{3}$$

Trace Anomaly Score: The Spectrum-based fault localization (SBFL) technique [25,26] has been widely used in software testing, which is mainly based on the coverage information of current program elements to assess the degree of program anomaly. Inspired by this, we calculate the anomaly score by using the SBFL technique, which is also mentioned in previous approaches [9,11,27,28].

The service is likely to be the root cause when the traces it goes through are more abnormal and fewer normal. Taking Figure 3 as an example, when most traces of $Req1$ are normal and most traces of $Req2$ are abnormal, we consider that ms_3 has a higher trace anomaly score than the other services. Each trace goes through multiple services. We count the e_f^i, e_p^i, and n_f^i of service i, where e_f^i is the numbers of abnormal traces containing service i, e_p^i is the numbers of normal traces containing service i, and n_f^i is the numbers of abnormal traces not containing service i. We calculate the trace anomaly score by using the Ochiai formula [29], which is popular in spectrum analysis. We use δ_{i2} to represent trace anomaly score. The range of δ_{i2} is $[0, 1]$. The trace anomaly score is defined as:

$$\delta_{i2} = \frac{e_f^i}{\sqrt{(e_f^i + n_f^i) \cdot (e_f^i + e_p^i)}}. \tag{4}$$

Service Correlation: Considering the impact of service dependencies, a larger service anomaly score does not necessarily indicate that the service is the root cause. Therefore, we need to further calculate service correlation to better analyze the process of anomaly propagation. We believe that when the metrics of two services are more correlated, it means that these two services are more likely affected by the same anomaly. In other words, the stronger the correlation, the higher the probability of anomaly propagation. Therefore, we use Pearson correlation [13,15,30] to calculate the metric correlation of the two abnormal services, i.e., the weight value W_{ij} when $e_{ij} = 1$. We use $corr_{ijk}$ to represent the correlation of the k-th metric between service i and service j. $corr_{ijk}$ is defined as:

$$corr_{ijk} = \left| \frac{\sum_{t=1}^{c}(M_{ik}^t - \overline{M_{ik}})(M_{jk}^t - \overline{M_{jk}})}{\sqrt{\sum_{t=1}^{c}(M_{ik}^t - \overline{M_{ik}})^2 \sum_{t=1}^{c}(M_{jk}^t - \overline{M_{jk}})^2}} \right|. \tag{5}$$

The range of $corr_{ijk}$ is $[0, 1]$. The larger the $corr_{ijk}$, the stronger the correlation between services. We choose the maximum value of the anomaly metric correlation as the initialized weight of the attributed graph edge. W_{ij} is defined as:

$$W_{ij} = \max\ corr_{ijk},\ k = 1, ..., m. \tag{6}$$

4.4 | Protection mechanism calibration

When the services turn on the circuit breaking or rate limiting mechanism, the way of anomaly propagate among services changes. Intuitively, the correlation between service metrics is reduced and the root cause location results are inaccurate. Therefore, we design a two-stage dynamic protection mechanism calibration strategy to eliminate the impact of the circuit breaking and rate limiting mechanisms.

When the circuit breaker is open, the correlation between services will reduce. When using correlation to locate the root causes, the circuit breaker returns the irrelevant result. To eliminate the effect of circuit breaker, we design the calibration of the circuit breaking mechanism, which includes three steps: status determination, metrics change, and correlation change. We determine the status of the circuit breaker by traces. We use in-neighbor service metrics to change the original metrics and thus replace the correlation when the circuit breaker is open. The circuit breaking mechanism calibration is described as follows.

Step1: Status determination. As shown in Figure 2, we use traces to determine the states of the circuit breaker. If consecutive k_1 requests are denied between service i and service j (the HTTP status code returned is $5XX$), the circuit breaker will be open. The duration opening time of the circuit breaker is KT seconds. After KT seconds, the circuit breaker goes into the half-open state. If the consecutive k_2 requests are normal, service i turns off the circuit breaker. Otherwise service i continues to turn on the circuit breaker. At this time, the duration opening time of the circuit breaker is $2 * KT$ seconds.

Step2: Metrics change: We get the time $[p, q]$ when the circuit breaker is open by step 1. We use the in-neighbor service set I_j to change the original metrics of service j. For each M_{jk}^t is collected during $[p, q]$, if M_{jk}^t records the response time, ${M_{jk}^t}' = \frac{\sum M_{rk}^t}{|I_j|}, r \in I_j$. If M_{jk}^t records the throughput, ${M_{jk}^t}' = \sum M_{rk}^t, r \in I_j$. Considering that the circuit breaker does not affect CPU usage and memory usage directly, we do not adjust these metrics.

Step3: Correlation change: Let the corrected ${M_{jk}^t}'$ to calculate the calibrated correlation score W_{ij}' between services.

The second stage is the rate limiting mechanism calibration. Intuitively, the more the number of requests, the greater the service correlation. But the rate limiting mechanism limits the number of requests and thus reduces the service correlation. Therefore, we need to eliminate the impact of rate limiting mechanism. We count the number of service requests rejected due to the rate limiting mechanism through the HTTP status of traces, calculate the impact factor of the rate limiting mechanism, and then correct the service correlation. The rate limiting mechanism calibration is described as follows.

Step4: Probability statistics. Based on the HTTP status of traces (set the status code returned by the rate limiting mechanism to 429), we can calculate the proportion θ_{ij} due to the rate limiting mechanism in a detection cycle. The larger the θ_{ij}, the greater the impact of the rate limiting mechanism on service correlation. θ_{ij} is defined as:

$$\theta_{ij} = \frac{Hsize_{ij}}{Tsize_{ij}}, \tag{7}$$

where $Hsize_{ij}$ denotes the number of rejected requests and $Tsize_{ij}$ denotes the total number of requests between service i and j.

Step5: Impact factor calculation. The greater the percentage of services rejected due to the rate limiting mechanism, the lower the correlation between services. Inspired by reliable strategy [31], we define the impact factor of the rate limiting mechanism on service correlation as:

$$RM_{ij} = e^{\frac{\theta_{ij}}{1+\theta_{ij}}}. \tag{8}$$

Step6: Correlation change. The anomaly metrics correlation is updated as:

$$W''_{ij} = RM_{ij} \cdot W'_{ij}. \tag{9}$$

According the presented steps, we use the corrected correlation score W''_{ij} to replace W_{ij} when the protection mechanisms are open. Please note that there are other microservice protection mechanisms. We only analyze two common protection mechanisms, including circuit breaking and rate limiting mechanisms. Other types of protection mechanisms can also be designed with specific calibration strategies to further improve the accuracy of root cause localization.

Algorithm 1 Dynamic protection mechanism calibration strategy

Input: metrics set $\mathbf{M}$, traces set T, k_1,k_2,KT, initialized correlation matrix $\mathbf{W}$;
Output: Calibrated correlation matrix $\mathbf{W}$;

1: **for** Each $e_{ij} \in E$ **do**
2: Obtain all T containing v_i and v_j;
3: Calculate the duration time $[p, q]$ by step 1 of the circuit breaking mechanism calibration;
4: Calculate the percentage θ_{ij} of rejected requests by Eq.7;
5: **for** each type of metric M_{jk} **do**
6: **for** each metric M^t_{jk} during $[p, q]$ **do**
7: **if** M^t_{jk} records response time **then**
8: $M^t_{jk} \longleftarrow \frac{M^t_{rk}}{|I_j|}, r \in I_j$
9: **end if**
10: **if** M^t_{jk} records throughput **then**
11: $M^t_{jk} \longleftarrow \sum M^t_{rk}, r \in I_j$
12: **end if**
13: **end for**
14: **end for**
15: Calculate service correlation using the corrected metrics by Eq.6;
16: Calculate rate limiting impact factor by Eq.8;
17: Update W_{ij} by Eq.9;
18: **end for**
19: Return calibrated correlation matrix $\mathbf{W}$.

Algorithm 1 describes the process of the dynamic protection mechanism calibration strategy. The time complexity of the strategy is related to the number n of services, the number N of traces, the type number m of metrics, and the collected number c of each metric during a detection cycle. Obtaining all T containing v_i and v_j in Step 2 can be done in $O(Nn^2)$ time. Calculating the time when the circuit breaker is open in Step 3 and the probability that the requests are rejected due to the rate limiting mechanism in Step 4 can be performed in $O(Nn^2)$ time. The time complexity of adjusting origin metrics in Step 5-14 is $O(cmn^2)$. Calculating the metric correlation in Step 15 can be done in $O(cn^2)$ time. Calculating the rate limiting impact factor in Step 16 and Updating W_{ij} in Step 17 can be performed in $O(n^2)$ time. Overall, the time complexity of the dynamic protection mechanism calibration strategy is $O(cmn^2 + Nn^2)$.

4.5 | Abnormal services ranking

In this section, we use the PageRank algorithm to locate root cause of abnormal services, which has proven good performance in anomaly propagation and root cause localization [11,13,32,33]. The PageRank algorithm is a well-known approach for web analysis that aims to rank the importance of web pages. Similarly, we can iterate over the anomaly attributed graph $G(V, E)$ to calculate the probability of the root causes. When $e_{ij} \in E$, we define the probability of anomaly propagation as W_{ij}. We use W_{ij} to denote the corrected W''_{ij} for easy description.

Inspired by MonitorRank [34], it is easy for a walker to enter a "trap" when the walker keeps moving forward. When the current service has little correlation to its out-neighbor services, the walker will perform no other actions. In order to make the correlation more inspiring, we design the backward propagation path. When $e_{ij} \in E$ and $e_{ji} \notin E$, we consider that the anomaly has $\rho_1 W_{ij}$ probability of being passed from service j to service i. A larger value of ρ_1 indicates more flexible path selection for the walker. The probability of backward propagation is defined as:

$$W_{ji} = \rho_1 W_{ij}, e_{ij} \in E \text{ and } e_{ji} \notin E, \tag{10}$$

where ρ_1 denotes the backoff coefficient (default $\rho_1 = 0.4$ in this paper).

We consider the walker stays longer at the current service when the correlation scores are low between its in-neighbor and out-neighbor services. We use the anomaly scores δ_{i1} and δ_{i2} to calculate the average anomaly score S_i of service i. Considering that the value range of δ_{i1} is not $[0, 1]$, we normalize the δ_{i1}, namely δ'_{i1}. We consider both anomaly scores equally important. Thus, S_i is defined as:

$$S_i = \frac{\delta'_{i1} + \delta_{i2}}{2}. \tag{11}$$

The probability of W_{ii} is equal to the service anomaly score S_i subtracted by the maximum correlation of the in-neighbor services I_i and the out-neighbor services O_i. W_{ii} is define as:

$$W_{ii} = \max(0, \rho_2 S_i - \max(W_{ji}, W_{ir})), j \in I_i \text{ and } r \in O_i, \tag{12}$$

where ρ_2 denotes the self coefficient (default $\rho_2 = 0.7$ in this paper).

We define the transition probability matrix $\mathbf{P}$ based on $\mathbf{W}$ as:

$$P_{ij} = \frac{W_{ij}}{\sum_j W_{ij}}. \tag{13}$$

Moreover, the current service is influenced by other services if it contains incoming and outcoming abnormal invocations[15,20]. In order to calculate the additional teleportation vector U more accurately in the PageRank algorithm, we calculate the suspicious score by the absolute value of the difference between the numbers of traces that contain incoming abnormal invocations Of_{iin} and outcoming abnormal invocations Of_{iout}. The suspicious score is defined as:

$$IS_i = |Of_{iout} - Of_{iin}|. \tag{14}$$

After that we use IS_i and anomaly scores S_i to calculate the additional teleportation vector $U = \{u_1, u_2, \cdots, u_n\}$. u_i is defined as:

$$u_i = \frac{IS_i \cdot S_i}{\sum_{i=1}^{n}(IS_i \cdot S_i)}. \tag{15}$$

Then we use the PageRank algorithm to calculate the root causes by Equation 1. We set the starting round $X_0 = [1/n, ..., 1/n]$. When the difference between the previous round and the current round is less than ε, we can get the ranking results X of the abnormal services.

Algorithm 2 The random walking approach

Input: Service attributed graph $G(V, E)$, service anomaly score set δ'_1 and δ_2, service traces set T, correlation matrix $\mathbf{W}$;
Output: Microservice ranking results set X;

for Each $v_i, v_j \in V$ **do**
  Calculate S_i, S_j by Eq.11;
  if $i == j$ **then**
    Calculate W_{ii} by Eq.12;
  end if
  if $e_{ij} \in E$ and $e_{ji} \notin E$ **then**
    Calculate W_{ji} by Eq.10;
  end if
end for
for Each $v_i \in V$ **do**
  Obtain all T containing v_i;
  Calculate the suspicious score IS_i by Eq.14;
end for
Calculate the iteration matrix $\mathbf{P}$ by Eq.13, the additional teleportation vector U by Eq.15;
while $|X_{cur} - X_{pre}| > \varepsilon$ **do**
  $X_{pre} \leftarrow X_{cur}$;
  Calculate X_{cur} by Eq.1;
end while
Return service root cause ranking results X.

In Algorithm 2, we describe the process of random walking approach. The time complexity of the random walking approach is related to the number of services n, the number of traces N, and the iteration threshold ε. Calculating the anomaly scores and updating W_{ij} in Step 1-9 can be done in $O(n^2)$ time. Computing the suspicious score in Step 10-13 can be performed in $O(Tn)$ time. Calculating the iteration matrix and additional teleportation vector in Step 14 can be done in $O(n^2)$ time. The time complexity of the PageRank algorithm in Step 15-18 is $O(t(\varepsilon) \cdot n^2)$ time[35], where $t(\varepsilon)$ is the number of iterations. Overall, the time complexity of the random waking approach is $O(Tn + t(\varepsilon) \cdot n^2)$.

5 | EVALUATION

5.1 | Experiment setup

We evaluate the performance and efficiency of MicroGBPM in a distributed computing platform, which has 5 physical nodes. Each node has a 4-core CPU, 64 GB memory and runs with Centos7.6 operating system. Gigabit switch network is used for communication between the nodes. The containers are managed by using Kubernetes (version 1.16.3) technology. The platform has been deployed with open source tools such as APISIX, Istio, Prometheus, OpenTracing, etc.

We deploy two open microservice systems and obtain metrics and traces separately to evaluate MicroGBPM. Train-Ticket system provides typical train ticket booking related functions such as ticket inquiry, ticket reservation, payment, ticket change, user notification, etc. It uses a total of four programming languages: Java, Python, Node.js, and Go. The Bookinfo application is divided into four separate microservices, including ProductPage, Reviews, Ratings, and Details. This application mimics a category in an online bookstore and displays information of books, such as ISBN, number of pages, and some reviews. We set the detection period to 5 minutes and collect metrics per 10s. We use Jmeter[#] to simulate 20 requests per second and collect traces by OpenTracing.

We inject three kinds of service failures to analyze MicroGBPM, including application bugs, network latency, and CPU exhaustion. We use APISIX to inject different faults into the services. According to Occam's razor theory[36], the probability of a complex system with two root causes at the same time is very low. We inject at most two service failures at the same time. We inject each type of fault 20 times in different services. Considering the effect of randomness and different parameters, we repeat each injection 5 times. The time of each injection lasts 2 minutes.

We use APISIX to introduce the protection mechanisms. We configure different k_1, k_2, and KT in the api-breaker plugin of APISIX. We implement the rate limiting mechanism with the leaky bucket limiter of APISIX and set the maximum request rate and the returned HTTP status code 429. In the experiment, we set different values of k_1, k_2, KT and calculate θ_{ij} to analyze the function of the protection mechanism calibration module.

5.2 | Evaluation metrics and baselines

In order to quantitatively evaluate the accuracy of MicroGBPM, we use $T@k$ and the Avg Score for evaluation.

Top-k($T@k$) accuracy: The probability that the root causes are in the top k services. The larger the $T@k$, the more accurate result of root cause localization. We consider $k = 1, 2, 3$ in one cause experiments and $k = 2, 3, 5$ in two causes experiments. Let r_i be the root causes of the $i\text{-}th$ anomaly and $Rank_i^j$ be the $j\text{-}th$ top result of $i\text{-}th$ anomaly. When $Rank_i^j \in r_i$, $t_i^j = 1$. Otherwise, $t_i^j = 0$. $T@k$ is defined of a set of given anomalies A as:

$$T@k = \frac{1}{|A|} \sum_{i=1}^{|A|} \frac{\sum_{j=1}^{k} t_i^j}{\min(k, |r_i|)}. \tag{16}$$

Average Score: The Avg Score refers to the average of all root cause microservice ranks, which describes the overall performance of the algorithm. When there is one root cause, the Avg Score is the average of $T@1$, $T@2$, and $T@3$. When there are two root causes, the Avg Score is the average of $T@2$, $T@3$, and $T@5$.

To better evaluate the effectiveness of MicroGBPM, we compare it with T-Rank[9], TraceRank[11], Microscope[10], MicroRCA[13], and MicroHECL[15] approaches respectively. T-Rank utilizes a lightweight approach based on spectrum analysis to infer the root causes. We use the Ochiai spectrum methods based on the collected traces to compare with T-Rank. To compare with TraceRank, we combine the spectrum analysis and the random walking algorithm to locate the root causes. Microscope constructs the service causal graph by using metrics. It then uses causal relationship and traverses the causal graph to locate the root cause. We use collected metrics and cause graph to compare with Microscope. MicroRCA is based on the attributed graph that models the propagation of anomalies between services. To compare with MicroRCA, we use metrics to construct attributed graphs and locate the root causes by the personalized PageRank approach. MicroHECL uses the service call to construct the service dependency graph and analyzes the way of anomaly propagation. To compare with MicroHECL, we locate the root causes by caculating the correlation of target services and downstream services based on the service dependency graph.

[#] https://jmeter.apache.org/

5.3 | Effectiveness of anomaly detection module

To evaluate the effectiveness of the anomaly detection module, we count the number tp of true positives, the number fn of false negatives, and the number fp of false positives. We calculate $precision$, $recall$, and $f1$ scores, where $precision = \frac{tp}{tp+fp}$, $recall = \frac{tp}{tp+fn}$, and $f1 = 2 \times \frac{precision \times recall}{precision+recall}$. From the Figure 5, we can see that MicroGBPM achieves high $precision$, $recall$ and $f1$ scores in different systems. It means that the anomaly detection module can catch most of the anomalies. In addition, two root causes at the same time are easier to detect anomaly than one cause. The $precision$, $recall$, and $f1$ scores in TrainTicket system are higher than those in Bookinfo system when there exists the same number of root causes. Because it is easier to detect anomaly due to the complex topology of TrainTicket. Considering the more complex in the real-world systems, the anomaly detection module will perform better.

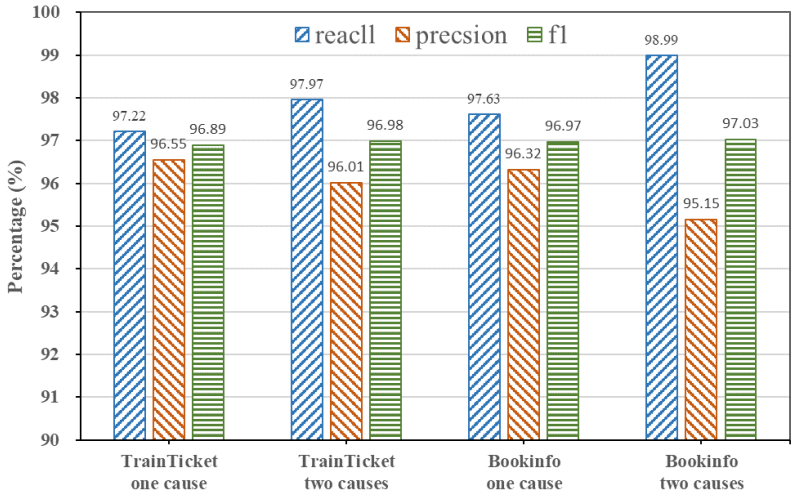

FIGURE 5 The anomaly detection results in the TrainTicket system and Bookinfo system

5.4 | Effectiveness of protection mechanism calibration module

5.4.1 | Overall effectiveness of the protection mechanism calibration module

The Figure 6 shows the difference of the Avg Score with and without the protection mechanism calibration module. With the introduce of the protection mechanism calibration module in MicroGBPM, the Avg Score increases, indicating that the protection mechanism calibration module plays an important role in root cause localization. It can effectively improve the accuracy of root cause localization. In our experiments, the Avg Score is increased about 7.2% when exists two causes in the TrainTicket system with protection mechanism calibration module. Since the Bookinfo system is relatively simple, the improvement of the protection mechanism calibration module is not significant. The more complex the system, the more obvious the improvement brought by the protection mechanism calibration module.

5.4.2 | Impacts of configuration related to the protection mechanisms

In Section 4, the parameters related to the protection mechanism calibration module are k_1, k_2, KT, and θ_{ij}. To eliminate the effect of different causes on the results, we only analyze the benchmark with two causes in the TrainTicket system. In this section, we use the control variables approach to analyze the impacts of each parameter and calculate the Avg Score to reflect the accuracy of root cause localization.

We set the maximum request rate 20 per second when evaluate the impact of different k_1, k_2, and KT on the root cause localization in the circuit breaking mechanism. We set $k_2 = 2$ and $KT = 5s$ when analyzing the impacts of k_1. From Figure 7, we can see that the MicroGBPM outperforms the other approaches overall due the impact of the circuit breaking mechanism. In our experiments, the Avg Score of MicroGBPM is increased about 16% compared with T-Rank when $k_1 = 2$. As k_1 increases, the overall accuracy gets better as shown in Figure 7. Because the effect of the circuit breaking mechanism on service correlation will reduce gradually. In addition, we observe that the smaller the k_1, the larger the gap between MicroGBPM and other approaches. It indicates that MicroGBPM can effectively improve the accuracy of root cause localization when the circuit breaker is open. As the k_1 increases, the effect brought by the circuit breaker becomes smaller and smaller, and the superiority of

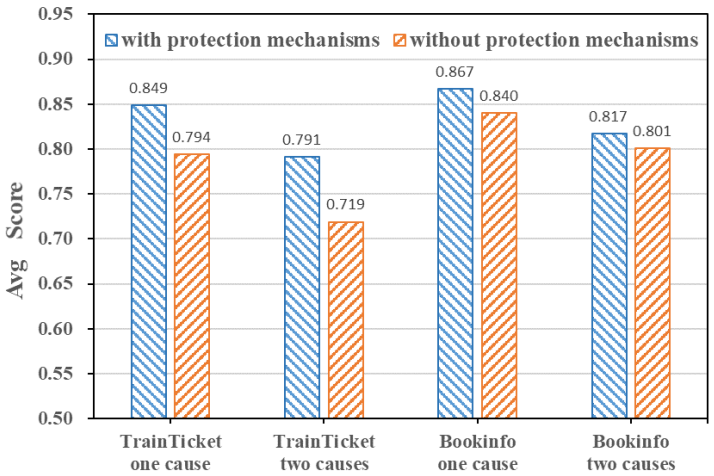

FIGURE 6 Comparison with and without the protection mechanism calibration module

MicroGBPM is not obvious. Similarly, we set $k_1 = 6$ and $KT = 5s$ when analyzing the impacts of k_2 and set $k_1 = 6$ and $k_2 = 2$ when analyzing the impacts of KT. When KT and k_2 increase, the duration opening time of the circuit breaker becomes longer and the closing conditions of the circuit breaker become more difficult, leading to the gradual decrease of the Avg Score as show in Figure 8 and Figure 9. MicroGBPM outperforms the other approaches overall. And the larger the KT and k_2, the larger the gap between MicroGBPM and other approaches, also indicating that MicroGBPM can effectively improve the accuracy in microservice systems with the circuit breaking mechanisms.

We evaluate the impact of different levels of rate limiting mechanism on the root cause localization when $k_1 = 6$, $k_2 = 2$, $KT = 5s$. We set different maximum request rates and count the overall percentage of traces affected by the rate limiting mechanism. We calculate the Avg Score to analyze the results of the root cause localization. As shown in Figure 10, we divide the percentage into 5 groups such as $< 5\%$, $5\% \sim 10\%$, $10\% \sim 15\%$, $15\% \sim 20\%$, and $> 20\%$. As the percentage increases, the correlation between services substantially reduces and the accuracy of each approaches gradually decreases. The reason is that the larger the percentage, the greater the probability that the requests are rejected, leading to the worse of the Avg Score. In addition, the larger the percentage, the more obvious advantage of MicroGBPM compared with other approaches, indicating that MicroGBPM can effectively improve the accuracy in microservice systems with the rate limiting mechanism.

In summary, the accuracy of MicroGBPM under the impact of the protection mechanisms is better than other approaches. The greater the impact of the protection mechanisms, the more obvious advantage of MicroGBPM.

5.5 | Effectiveness of the root cause localization

In this section, we analyze the effectiveness of root cause localization when inject different types of faults and the comparison with other state-of-art approaches respectively. Note that this section focuses on the overall results. Each type of fault in each benchmark has the same configuration.

5.5.1 | Impacts of different types of faults

The Table 2 and Table 3 show the accuracy of different types of faults in two microservices systems with one root cause and two root causes. We can see that the accuracy of application bug is the highest among all benchmarks. In addition, the accuracy of one root cause is higher than that of two root causes. The reason is that the anomalous behavior of the system is more serious when two faults are injected at the same time, and it is more difficult to locate two root causes accurately. The accuracy of MicroGBPM in the Bookinfo system is higher than that in the TrainTicket system. Considering the simple structure of the Bookinfo system, it is easier to locate the root causes of abnormal services accurately.

5.5.2 | Comparisons with other state-of-art approaches

We compare the other approaches separately as shown in the Table 4 and Table 5. We can see that the accuracy of MicroGBPM outperforms the other approaches in one cause or two causes systems. The Avg Score of MicroGBPM is increased about 10% compared with T-Rank in TrainTicket system with one cause. This is because T-Rank only uses spectrum analysis to calculate the score of service anomalies. Microscope uses the causal graph constructed by metrics and does not make use of traces. The Avg Score of MicroGBPM is increased about 11.8% compared with Microscope in TrainTicket system with one cause. MicroHECL uses the service call relationship to locate the root causes without considering the whole anomaly

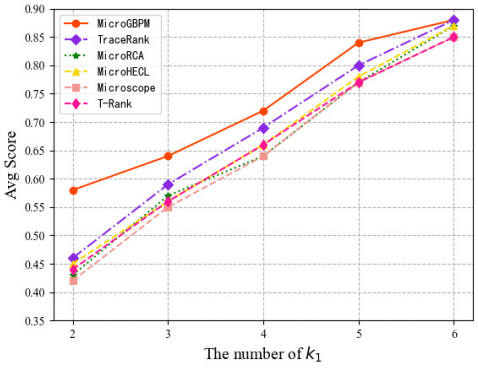

FIGURE 7 The impacts of k_1

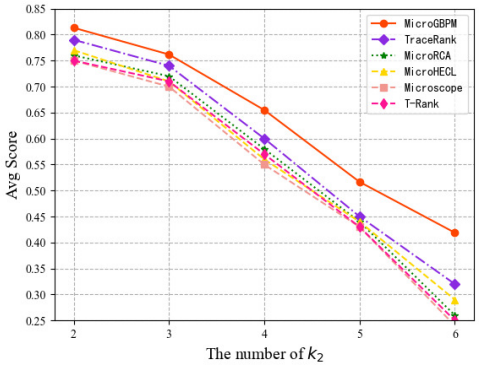

FIGURE 8 The impacts of k_2

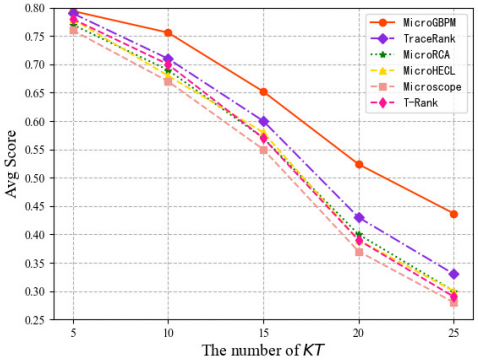

FIGURE 9 The impacts of KT

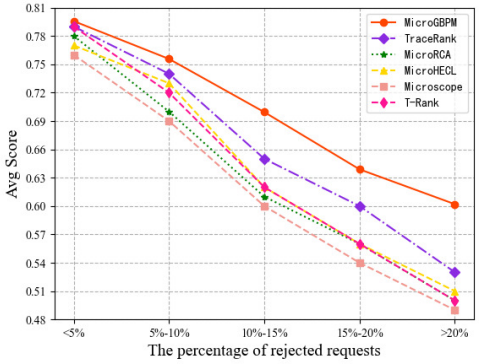

FIGURE 10 The impacts of the rate limiting mechanism

TABLE 2 Overall effectiveness evaluation of different types of faults with one cause

Benchmarks	Fault type	$T@1$	$T@2$	$T@3$	Avg Score
TrainTicket one cause	Application bug	0.824	0.879	0.923	0.875
	Network latency	0.734	0.840	0.915	0.830
	CPU exhaustion	0.800	0.853	0.874	0.842
BookInfo one cause	Application bug	0.840	0.883	0.968	0.897
	Network latency	0.750	0.854	0.917	0.840
	CPU exhaustion	0.806	0.867	0.918	0.864

TABLE 3 Overall effectiveness evaluation of different types of faults with two causes

Benchmarks	Fault type	$T@2$	$T@3$	$T@5$	Avg Score
TrainTicket two causes	Application bug	0.742	0.804	0.856	0.801
	Network latency	0.745	0.809	0.840	0.798
	CPU exhaustion	0.694	0.786	0.847	0.776
BookInfo two causes	Application bug	0.796	0.827	0.878	0.833
	Network latency	0.758	0.818	0.848	0.808
	CPU exhaustion	0.773	0.804	0.856	0.811

propagation process. Thus, the Avg Score of MicroGBPM is increased about 8.6% compared with MicroHECL in TrainTicket system with one cause. TraceRank and MicroRCA are similar to our approach. But none of them takes into account the impact of the protection mechanisms on the root cause localization. Therefore, the Avg Score of MicroGBPM is increased about 6.6% compared with TraceRank and 10.1% compared with MicroRCA in TrainTicket system with one cause. In summary, the accuracy of MicroGBPM is better than the other approaches.

TABLE 4 Overall effectiveness evaluation of one root cause compared with other approaches (best scores are in boldface)

Benchmarks	Approach	$T@1$	$T@2$	$T@3$	Avg Score
TrainTicket one cause	MicroGBPM	**0.786**	**0.857**	**0.904**	**0.849**
	T-Rank	0.711	0.757	0.814	0.746
	TraceRank	0.732	0.804	0.864	0.783
	Microscope	0.689	0.732	0.814	0.731
	MicroRCA	0.696	0.761	0.829	0.748
	MicroHECL	0.718	0.768	0.846	0.763
Bookinfo one cause	MicroGBPM	**0.799**	**0.868**	**0.934**	**0.867**
	T-Rank	0.701	0.830	0.865	0.799
	TraceRank	0.736	0.837	0.931	0.834
	Microscope	0.694	0.799	0.865	0.786
	MicroRCA	0.712	0.813	0.882	0.802
	MicroHECL	0.708	0.830	0.889	0.809

TABLE 5 Overall effectiveness evaluation of two root causes compared with other approaches (best scores are in boldface)

Benchmarks	Approach	$T@2$	$T@3$	$T@5$	Avg Score
TrainTicket two causes	MicroGBPM	**0.727**	**0.799**	**0.848**	**0.791**
	T-Rank	0.640	0.678	0.734	0.684
	TraceRank	0.668	0.720	0.761	0.716
	Microscope	0.588	0.637	0.702	0.642
	MicroRCA	0.661	0.682	0.744	0.696
	MicroHECL	0.657	0.692	0.751	0.700
Bookinfo two causes	MicroGBPM	**0.776**	**0.816**	**0.861**	**0.817**
	T-Rank	0.673	0.782	0.840	0.765
	TraceRank	0.711	0.820	0.857	0.796
	Microscope	0.650	0.782	0.847	0.760
	MicroRCA	0.701	0.796	0.850	0.782
	MicroHECL	0.711	0.813	0.850	0.791

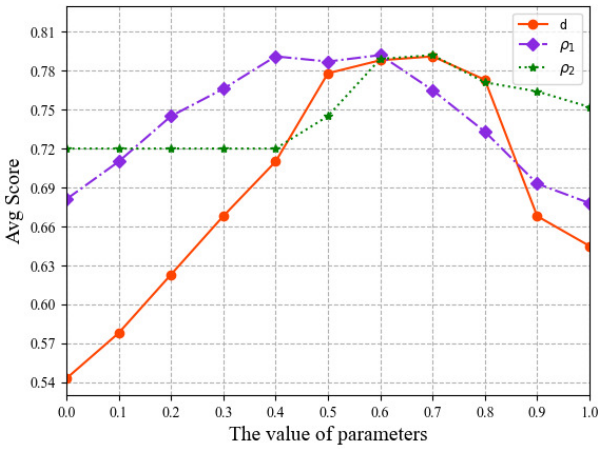

FIGURE 11 The impacts of different configuration on the Avg Score

5.6 | Impacts of configuration

In this section, we analyze the impact of d, ρ_1, and ρ_2 on the results of root cause localization with two root causes in the TrainTicket system. We consider the case of $k_1 = 6$, $k_2 = 2$, $KT = 5s$, and the maximum request rate 20 per second. We use the Avg Score to reflect the overall accuracy of MicroGBPM. As shown in the Figure 11, the x axis presents the value of parameters, while the y axis presents the Avg Score.

The damping parameter d in Equation 1. The damping parameter d is introduced in the PageRank algorithm. The Avg Score exhibits a convex shape as d increases. We find the reason is that the walker no longer wanders when the $d = 0$. It is difficult to locate the two root causes accurately at this time. The result of root cause localization will be more accurate as d increases. However, when $d > 0.7$, it is difficult for the walker to avoid termination services (the out-degree is 0) and trap services (the service only calls itself), resulting in the decrease of the Avg Score.

The fallback coefficient ρ_1 in Equation 10. In this paper, the fallback coefficient ρ_1 is introduced to increase the flexibility of the walker. If the current service is less relevant to the out-neighbor services, the walker can back off. The Avg Score shows a convex shape as ρ_1 increases. We find the reason is that as ρ_1 increases, the walker becomes more flexible and the results of root cause localization become more accurate. When $\rho_1 = 0.4$, the Avg Score is highest. However, when $\rho_1 > 0.6$, the walker will backtrack even if the current service is strongly correlated with its out-neighbor services. In this case, it is difficult to locate the root causes accurately, resulting in the decrease of the Avg Score.

The self coefficient ρ_2 in Equation 12. When the service is not correlated with both in-neighbor and out-neighbor services, the walker should stay at the current service longer. Thus, ρ_2 is introduced in this paper. When ρ_2 is small, the calculated W_{ii} is always 0 and the Avg Score remains constant. After that, the Avg Score shows a convex shape as ρ_2 increases. We find the reason is that as ρ_2 becomes larger, the probability that the service will stay becomes higher and higher and the results becomes more accurate. When $\rho_2 = 0.7$, the Avg Score is highest. However, the larger service anomaly scores are not necessarily to the root causes. It will lead to the decrease of the Avg Score when $\rho_2 > 0.7$.

6 | RELATED WORK

In recent years, with the rapid development of the Internet, microservice systems have become more and more complex. To guarantee the quality of service operation, microservice root cause localization has been widely studied, including trace analysis and service dependency graph. The approaches of service dependency use causal analysis or service calls to locate the root causes. Although the existing approaches have studied the problem of locating the root cause of abnormal services, it is not suitable for microservice systems with protection mechanisms. In addition, different levels of protection mechanisms have different effects on root cause localization. The impact of protection mechanisms on root cause localization has not been sufficiently considered.

The trace analysis approaches use metrics and traces to calculate the degree of anomalies of services and prioritize the localization of services with a high degree of anomalies. Zhou et al.[7] proposed MEPFL, which was an approach for potential error prediction and abnormal localization of microservice applications. It collects service metrics and then trains the prediction model at traces level and services level to predict service faults. Hou et al.[8] proposed Diagnose, which used heterogeneous data source to jointly diagnose faults in microservice systems. Diagnose quickly locates the abnormal service in the channel through unsupervised algorithms and voting-based location strategies. Ye et al.[9] proposed T-Rank that used

a spectrum analysis algorithm to locate root causes of abnormal services. Yu et al.[27] proposed MicroRank. It firstly differentiates traces, then uses the PageRank algorithm to analyze the importance of different traces, and finally uses spectrum analysis to locate root cause of abnormal services. Li et al.[20] proposed TraceRCA approach, which considered that microservices with more abnormal traces passing through were more likely to be the root cause. It uses incoming and outgoing abnormal traces to infer the anomaly propagation pattern and calculates the suspicious score.

The causal analysis approaches use the correlation between service metrics to determine the service dependencies and then simulate the walking process to calculate the root causes. Lin et al.[10] proposed a new system called Microscope to identify and locate anomalous services. Microscope uses service metrics to construct service causal graphs and infers the cause of performance problems in real time. Ma et al.[12] proposed the MS-Rank root cause localization framework. MS-Rank uses the service metrics to construct the service dependency graphs. It then simulates the random walking process and locates the service root cause. Meng et al.[37] presented the MicroCause framework. It uses a combination of the path conditional time series PCTS algorithm and the random walking TCORW algorithm to locate root cause of abnormal services. Aggarwal et al.[38] described a lightweight fault localization system, which built causal relationships with metrics and logs and further leveraged PageRank centrality of the derived causal graph for generating a ranked list of abnormal microservices.

The approaches based on service call calculate the root cause by traversing the service call graph. Wu et al.[13] proposed the MircroRCA system. It uses attributed graphs that model the propagation of anomalies between services and machines to locate faults. Wang et al.[39] proposed a statistical-based approach for automatic fault diagnosis of microservices. It develops trace baseline algorithm through call trees and then uses tree edit distance and principal component analysis to locate anomalous services. Kim et al.[40] proposed the MonitorRank algorithm to find the root cause of abnormal services by service call graphs and service metrics. Liu et al.[15] proposed the MicroHECL approach. It analyzes possible anomaly propagation chains based on dynamically constructed service graphs and ranks candidate service root cause causes based on correlation analysis.

However, considering the impact of protection mechanisms, the above approaches cannot accurately locate the root cause of abnormal services. Unlike previous approaches, when the service response time is abnormal, we build abnormal service attributed graphs and eliminate the impact of protection mechanisms to locate the root causes of the abnormal services.

7 | CONCLUSION

In this paper, we propose a new framework named MicroGBPM to locate the root cause of abnormal services, considering the impact of protection mechanisms. We initiate the process of root cause location when the service response time is abnormal. Firstly, we construct a service attributed graph by using metrics and traces. We calculate the service anomaly scores and the correlation of services. In addition, we design a two-stage dynamic calibration strategy to eliminate the impact of the protection mechanisms. Then we propose a random walking approach to obtain the ranking results of the root causes. The experimental results show that MicroGBPM improves the accuracy of root cause localization compared to other approaches in the microservice systems with protection mechanisms.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Plan under Grant No. 2020YFF0305501, and the National Natural Science Foundation of China (NSFC) under Grant No. 62172050.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Haitao Zhang https://orcid.org/0000-0002-9131-3517

References

1. Zhou X, Peng X, Xie T, et al. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering* 2018; 47(2): 243–260.

2. Di Francesco P, Malavolta I, Lago P. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In: 2017 IEEE International Conference on Software Architecture (ICSA). IEEE. ; 2017: 21–30.

3. Alshuqayran N, Ali N, Evans R. A systematic mapping study in microservice architecture. In: 2016 IEEE 9th international conference on service-oriented computing and applications (SOCA). IEEE. ; 2016: 44–51.

4. Cerny T, Donahoo MJ, Trnka M. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review* 2018; 17(4): 29–45.

5. Soldani J, Montesano G, Brogi A. What Went Wrong? Explaining Cascading Failures in Microservice-Based Applications. In: Symposium and Summer School on Service-Oriented Computing. Springer. ; 2021: 133–153.

6. Kelly S. Estimating economic loss from cascading infrastructure failure: a perspective on modelling interdependency. *Infrastructure Complexity* 2015; 2(1): 1–13.

7. Zhou X, Peng X, Xie T, et al. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In: 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM. ; 2019: 683–694.

8. Hou C, Jia T, Wu Y, Li Y, Han J. Diagnosing Performance Issues in Microservices with Heterogeneous Data Source. In: 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). IEEE. ; 2021: 493–500.

9. Ye Z, Chen P, Yu G. T-Rank: A Lightweight Spectrum based Fault Localization Approach for Microservice Systems. In: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid). IEEE. ; 2021: 416–425.

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

11. Yu G, Huang Z, Chen P. TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. *Journal of Software: Evolution and Process* 2021: e2413.

12. Ma M, Lin W, Pan D, Wang P. Self-Adaptive Root Cause Diagnosis for Large-Scale Microservice Architecture. *IEEE Transactions on Services Computing* 2022; 15(3): 1399-1410.

13. Wu L, Tordsson J, Elmroth E, Kao O. MicroRCA: Root cause localization of performance issues in microservices. In: 2020 IEEE/IFIP Network Operations and Management Symposium(NOMS). IEEE. ; 2020: 1–9.

14. Guo X, Peng X, Wang H, et al. Graph-based trace analysis for microservice architecture understanding and problem diagnosis. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM. ; 2020: 1387–1397.

15. Liu D, He C, Peng X, et al. MicroHECL: high-efficient root cause localization in large-scale microservice systems. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE. ; 2021: 338–347.

16. Jeh G, Widom J. Scaling personalized web search. In: Proceedings of the 12th international conference on World Wide Web. ACM. ; 2003: 271–279.

17. Tibshirani, Robert and Walther, Guenther and Hastie, Trevor . Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 2001; 63(2): 411–423.

18. Ahmed T. Online anomaly detection using KDE. In: IEEE Global Telecommunications Conference. IEEE. ; 2009: 1–8.

19. Meloche J. Asymptotic behaviour of the mean integrated squared error of kernel density estimators for dependent observations. *The Canadian Journal of Statistics/La Revue Canadienne de Statistique* 1990; 18(3): 205–211.

20. Li Z, Chen J, Jiao R, et al. Practical root cause localization for microservice systems via trace analysis. In: 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS). IEEE. ; 2021: 1–10.

21. Ahmed F, Erman J, Ge Z, Liu AX, Wang J, Yan H. Detecting and localizing end-to-end performance degradation for cellular data services based on tcp loss ratio and round trip time. *IEEE/ACM Transactions on Networking* 2017; 25(6): 3709–3722.

22. Xu J, Wang Y, Chen P, Wang P. Lightweight and adaptive service api performance monitoring in highly dynamic cloud environment. In: 2017 IEEE International Conference on Services Computing (SCC). IEEE. ; 2017: 35–43.

23. Jayathilaka H, Krintz C, Wolski R. Performance monitoring and root cause analysis for cloud-hosted web applications. In: Proceedings of the 26th International Conference on World Wide Web. ACM. ; 2017: 469–478.

24. Lin F, Muzumdar K, Laptev NP, Curelea MV, Lee S, Sankar S. Fast dimensional analysis for root cause investigation in a large-scale service environment. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2020; 4(2): 1–23.

25. Kay S, Marple S. Spectrum analysis—A modern perspective. *Proceedings of the IEEE* 1981; 69(11): 1380-1419.

26. Cooper FS. Spectrum analysis. *The Journal of the Acoustical Society of America* 1950; 22(6): 761–762.

27. Yu G, Chen P, Chen H, et al. MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. In: Proceedings of the Web Conference. ACM. ; 2021: 3087–3098.

28. Zhang M, Li Y, Li X, et al. An empirical study of boosting spectrum-based fault localization via pagerank. *IEEE Transactions on Software Engineering* 2019; 47(6): 1089–1113.

29. Abreu R, Zoeteweij P, Van Gemund AJ. An evaluation of similarity coefficients for software fault localization. In: 2006 12th Pacific Rim International Symposium on Dependable Computing. IEEE. ; 2006: 39–46.

30. Benesty J, Chen J, Huang Y, Cohen I. *Pearson Correlation Coefficient*. Berlin, Heidelberg: Springer Berlin Heidelberg . 2009. ISBN 978-3-642-00296-0.

31. Liu Z, Fan G, Yu H, Chen L. Modelling and analysing the reliability for microservice-based cloud application based on predicate Petri net. *Expert Systems* 2021; 39(6): e12924.

32. Gleich DF. PageRank beyond the Web. *siam REVIEW* 2015; 57(3): 321–363.

33. Andersen R, Chung F, Lang K. Local partitioning for directed graphs using pagerank. In: International Workshop on Algorithms and Models for the Web-Graph. Springer. ; 2007: 166–178.

34. Kim M, Sumbaly R, Shah S. Root cause detection in a service-oriented architecture. *ACM SIGMETRICS Performance Evaluation Review* 2013; 41(1): 93–104.

35. Page L, Brin S, Motwani R, Winograd T. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab; : 1999. Previous number = SIDL-WP-1999-0120.

36. Jones JA, Harrold MJ, Stasko J. Visualization of test information to assist fault localization. In: Proceedings of the 24th International Conference on Software Engineering(ICSE). IEEE. ; 2002: 467–477.

37. Meng Y, Zhang S, Sun Y, et al. Localizing failure root causes in a microservice through causality inference. In: 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS). IEEE. ; 2020: 1–10.

38. Aggarwal P, Nagar S, Gupta A, et al. Causal Modeling based Fault Localization in Cloud Systems using Golden Signals. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD). IEEE. ; 2021: 124–135.

39. Wang T, Zhang W, Xu J, Gu Z. Workflow-aware automatic fault diagnosis for microservice-based applications with statistics. *IEEE Transactions on Network and Service Management* 2020; 17(4): 2350–2363.

40. Kim M, Sumbaly R, Shah S. Root cause detection in a service-oriented architecture. *ACM SIGMETRICS Performance Evaluation Review* 2013; 41(1): 93–104.