MFRL-CA: Microservice Fault Root Cause Location based on Correlation Analysis

YuHua Chen
cyhsikadeer@qq.com

NingJiang Chen*
chnj@gxu.edu.cn

WenXiu Xu
15866786838@163.com

LinMing Lian
394229081@qq.com

Huan Tu
2453742080@qq.com

College of Computer and Electronics Information, Guangxi University, Nanning, China

***Abstract*—With the expansion of the scale of microservice applications, its dynamic update and complex dependency call relations increase the probability of failure and the difficulty of diagnosis. How to ensure the stable operation of applications and QoS is crucial. At present, there are some researches on fault diagnosis of microservices, but there are still deficiencies in fault propagation and root cause localization. In this paper, a Microservice Fault Root cause Location Method Based on Correlation Analysis (MFRL-CA) is designed to reduce the time consumption of fault detection and root positioning. This method constructs the Microservice Fault Propagation Graph (MFPG) by collecting the correlation between dependent call data and historical fault data, and accurately infers and locates the fault causes through a new anomaly score measurement and random walk algorithm. Experiments show that this method can effectively detect faults and accurately locate the fault causes, and the accuracy is 12.75% higher than that of the benchmark method.**

Keywords: ***Microservice; Fault root cause location; Root cause analysis; Fault propagation graph***

I. INTRODUCTION

More and more enterprises adopt microservices as the development architecture of large-scale software systems. Microservices solve the problems of low flexibility, poor scalability and difficult change of traditional software systems. However, due to the complex dependence on call relationships and frequent delivery and deployment of microservices, it is difficult to diagnose microservice fault. In order to maintain the reliable operation and QoS of microservices, fault diagnosis of microservices is a subject of great research value[1].

The fault root location problem of microservice has always been the focus of fault diagnosis research of microservice. Microservice fault is more complex and difficult to locate than traditional software fault, and is different from traditional software fault. Literature [2] points out that about 45 % of microservice failures in 22 fault types are related to asynchronous invocation of service components, heterogeneous deployment and environment and configuration. There are three challenges in fault location of microservices. Firstly, there are many KPI indicators for microservice monitoring, and it is difficult to set the abnormal threshold of various indicators manually by operation and maintenance personnel. Literature [3] by means of service dependency and test analysis, we can infer the cause of microservice failure. In addition to monitoring QoS (Quality of Service) indicators, there are many monitoring indicators of microservice components and machine indicators of physical machines. Secondly, the fault propagation relationship of microservices is complex, and the operation and maintenance personnel describe the fault propagation relationship through the link tracking tool represented by Google Dapper[4] is not complete. Some existing research methods do not fully consider the failure relevance of the historical failure data of microservice[5]. Finally, in the microservice fault processing stage, a lot of time and manpower are needed to analyze, detect and locate the fault causes one by one. The operation and maintenance cost is relatively high and the efficiency is not high.

In response to the above problems, this paper constructs the MFPG by studying the microservice fault relationship and propagation path, and optimizes the traditional abnormal correlation method and random walk algorithm, and proposes a MFRL-CA. The main contributions are as follows.

(1) The dependency invocation relationship and service execution path of microservice are studied. According to microservice dependent call data and failure data, the Link Call Graph of end-to-end Tracking (LCGT) and the Microservice Fault Correlation Directed Graph (FCDG) are designed, and the construct MFPG based on the above two fault propagation graphs to describe the microservice fault propagation relationship.

(2) In terms of fault root location, this paper optimizes the calculation method of abnormal correlation, analyzes the limitations of traditional random walk algorithm, and designs a random walk algorithm with three search directions. Finally, based on the above research, MFRL-CA is proposed. It improves the global, flexibility and accuracy of fault search scope.

(3) According to the above method, the MFRL-CA framework prototype is designed and verified by experiments. Experimental results show that this method can locate the root cause of fault quickly and accurately.

II. RELATED WORKS

A. Microservice execution trajectory and dependent call monitoring technology

In terms of microservice execution trajectory tracking, there are three main categories of current execution trajectory tracking tools and methods: monitoring methods based on implantation, interceptor and data collection. Through research and analysis, this paper can construct the dependency

2767-6684/21/$31.00 ©2021 IEEE
DOI 10.1109/DSA52907.2021.00018

of microservices through the interactive data of microservice components.

(1) Monitoring method based on implantation: The implant-based approach aims to obtain the request execution path data of the program by embedding the monitoring program into the application. In recent years, some code injection tools (such as AspectJ[6]) can be used to modify the source code of the application, and then dynamically implant the monitoring code into the specified location at the system runtime, with the advantages of flexible setting and fine-grained monitoring. The embedded monitoring method requires developers to master the internal structure and program code of the system to be able to invade the system for code modification, which limits the scope of application of the method.

(2) Interceptor-based monitoring method: The interceptor-based monitoring method records the execution process of the system by intercepting requests. Most of the link tracking systems, such as Dapper, Zipkin, Pinpoint, etc., use intercepting requests to invade the system, but the intrusion is much lower than that based on code implantation.

(3) Monitoring method based on data collection: The monitoring method based on data collection is to analyze the system execution log obtained by machine learning algorithm, mine the log template, and then establish the statistical reasoning model of dependencies. Literatures [7,8] collect system log and performance data of distributed software systems as tracking data. However, log processing is not a simple work. The accuracy of service dependency graph constructed by log data is closely related to the degree of log detail.

B. Methods for locating the root cause of microservice failures

The difficulty of microservice fault diagnosis is how to quickly locate the root cause of the fault among the numerous faulty microservice components. Literature [9] finds faults through correlation analysis, and faulty nodes with higher correlation are more likely to be the root cause of the fault. Literature [10] assumes that the stronger the correlation of the abnormal node, the more likely it is the root cause node, and then calculates the correlation coefficient of the fault node through the Pearson coefficient, and finally uses a custom discrimination method to locate the root cause of the fault. Literature [11] collects all service execution trajectories through path testing, and when a fault occurs, analyzes the deviation between the faulty path and the normal path to locate the location of the fault.

The fault root cause location method based on graph theory is widely used in the field of microservice faults. Literature [12] designed a data collection and anomaly detection framework, and inferred the root cause of the failure based on the random walk algorithm. However, the algorithm needs to rely on the service-dependent call topology and system domain knowledge, and its generalization is not strong. Literature [13] constructs a causal relationship diagram through the PC causality algorithm, and searches for the root cause of the fault based on the random walk algorithm. Literature [14] collects the communication data of microservice, uses the causality method to construct the influence diagram, and then uses the second-order random walk algorithm to locate the root cause of the microservice failure. However, the reasoning of the causality requires a lot of system resources, and it cannot affect. The location of the root cause of the back-end failure of the front-end service is not obvious.

The dynamic dependence of microservice on complex calling relationships and heterogeneous microservice operating environments makes it difficult to analyze the scope of influence, propagation channels, and root causes of failures when faced with microservice failures, which brings huge problems to the development of microservice. It is urgent to find a microservice fault root cause location method that can detect faults, determine the scope, and locate the root cause of the fault to improve the efficiency of operation and maintenance.

III. BACKGROUND AND MOTIVATION

A. Analysis of fault propagation relationship

The complex operation of distributed systems will increase the occurrence of uncertain faults, and it is difficult to reproduce in system testing, such as concurrent asynchronous errors, fault propagation, resource competition, environmental configuration and so on. The research shows that uncertain faults account for 15% -80% of software fault[15]. Microservice is also a category of distributed architecture, which is a more fine-grained and lightweight distributed architecture. When a fault occurs in the microservice system, the associated components also fail by relying on invocation or other means of transmission, resulting in a large-scale microservice cascading failure.

Therefore, how to automatically diagnose microservice faults, we first need to understand the manifestation of microservice faults and the relationship between propagation and diffusion. In the microservice experimental environment as shown in figure 1. First of all, the Docker container is deployed on two physical machines Machine1 and Machine2, where Nginx, User and MySQL are login modules of microservice, and Stress[16] is a performance test service instance on Machine2. The microservice on the same physical machine share resources of the physical machine. There is a relationship of resource competition between them. At the same time, monitoring tools are deployed on each microservice node to collect the KPI data of the microservice. In an execution request, the dependent invocation relationship of microservice is nginx→user→mysql, while the microservice component mysql on Machine2 has no direct invocation relationship with Stress. In the problem study, the CPU utilization of the container was improved through Stress, and the performance of the microservice system was tested to verify whether the performance of the system was affected, and the fault propagation relationship between them was analyzed.

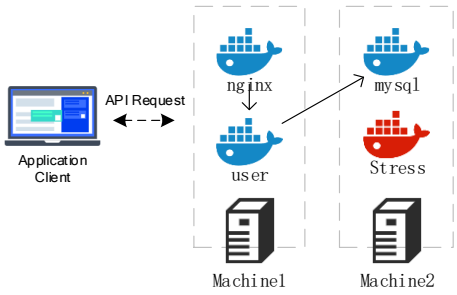

Figure 1. Microservice failure experiment environment

In the experiment, the CPU usage of Stress was gradually increased. Stress first experienced performance degradation, and then mysql performance declined, which eventually caused the microservice response time to time out. As shown in figure 2, from the time point start to increase the CPU usage, the high overhead of Stress begins to affect the response time of mysql on the same physical machine, and ultimately affects the entire microservice. Although Stress and microservices do not have a dependent calling relationship, microservice failures occur due to resource overhead. For this reason, there are two different types of failure propagation and diffusion methods for microservice failures: (1) Propagation through the dependent calling relationship of microservices abnormal; (2) Microservice components may cause other service failures due to failure correlation in the same service, node, location, etc.

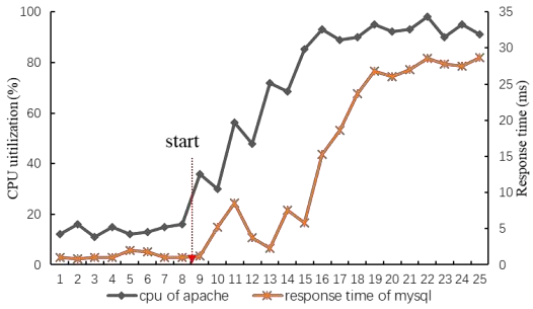

Figure 2. Changes in microservice monitoring indicators on the Machine2

B. *Correlation analysis of microservice KPIs*

When the microservice system is abnormal or faulty, the fault of the microservice component causes other components to be abnormal through the above two different fault propagation paths. When the service KPI index of microservice is abnormal, that is, when the KPI data are mutated, due to the delay of microservice fault, the machine KPI index related to abnormal often changes significantly before the fault. In the field of microservice fault, the abnormal degree of service can be measured by the correlation analysis of the change of microservice KPI. Literature [9] assumed that the more relevant KPI was to the abnormal index, the more likely it was to be the fault root. The Pearson coefficient was used to calculate the correlation between the front-end node and the back-end service node. The machine KPI with high correlation was more likely to be the fault root index. However, due to the "displacement" between the microservice KPIs, it was not necessarily effective to directly calculate the Pearson coefficient. However, there is a delay between different micro service nodes, resulting in a certain degree of "displacement" between KPIs of different micro service components. It is not accurate to directly calculate their Pearson coefficients.

In Table I, the correlation coefficient a and a∈[-1,1] of SS_TOTAL and CPU_PCT are calculated respectively. A =1 represents positive correlation, a=-1 represents negative correlation, and a=0 represents irrelevant. If the original KPI is directly used to calculate the correlation between them, it shows low or irrelevant; however, through the "translation" processing of the KPI time series, it shows a strong correlation.

TABLE I. USE PEARSON TO CALCULATE THE CORRELATION COEFFICIENT

Method	Translation	ss_total	CPU_pct
Pearson	True	0.1178	0.2164
	False	0.8365	0.8742

IV. OUR SOLUTION

This paper presents a method of MFRL-CA that simulating the fault detection and troubleshooting process of operation and maintenance engineers, it can quickly, accurately and automatically find faults, construct MFPG, locate and arrange the root cause indicators of faults. The MFRL-CA framework is shown in figure 3, which mainly includes three modules : Anomaly Detection, MFPG and Fault Root Localization.

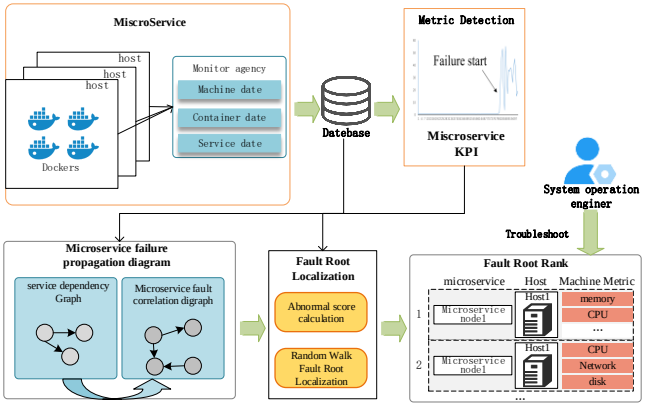

Figure 3. MFRL-CA framework architecture

(1) Anomaly detection: This module is mainly responsible for real-time monitoring of the service KPI and machine KPI of microservices, and judging whether the index is abnormal or faulty at a certain moment. After the microservice failure occurs, collect the microservice abnormal KPI data and trigger the microservice fault propagation diagram module and the fault root location module.

(2) Microservice fault propagation graph module: When a microservice fails, it constructs LCGT according to the execution trajectory of the microservice request, and constructs FCDG based on the fault association relationship. Finally, the two fault propagation graphs are constructed into a MFPG to describe the fault propagation diffusion

relationship, narrow the fault location range, and serve as the input of the fault root cause location module.

(3) Fault root location module: When the fault of the system is detected, the microservice KPI time series data and MFPG of the fault period are first input into the fault root location module. Then, the anomaly degree of each node in MFPG is calculated based on the anomaly scoring algorithm, so as to generate the transition probability matrix, and MFPG is accessed iteratively through the optimized random walk algorithm. Finally, Top-k possible fault root cause indicators are arranged through finite number of search visits. It realizes fast, accurate and automatic positioning of microservice fault root causes.

A. *Anomaly detection*

After the microservice fault occurs, it is necessary to detect the changes of KPIs of various machines before the microservice fault occurs. In order to detect the change time point of machine KPI, a CUSUM[17] change point detection algorithm based on sliding window is used to detect the machine KPI time series data before the failure, and accurately identify the start time of machine KPI change.

Through the above process, we can accurately mark the change time point of KPI time series data of various machines after the occurrence of microservice failure, and collect the time series data before and after the time point.

B. *Microservice fault propagation relationship graph*

In this paper, the method of MFPG can describe the two fault propagation paths of microservice according to the fault propagation relationship of microservice. This method first analyzes the dependency invocation relationship of microservice, then excavates the potential fault correlation of historical data of microservice, and analyzes entities such as services, components and containers that may be affected by faults from multiple dimensions. Finally, a MFPG is constructed to describe the fault range and propagation path, reduce the interference of other unrelated components on fault root location and reduce the loss caused by faults. In view of the complex dependence invocation relationship of microservice, graph structure can effectively obtain the dependence relationship between services, describe the manifestation and propagation path of service failure[18].

Related definitions: LCGT, FCDG and MFPG are defined as follows.

Definition 1: LCGT, labeled as $G_l = (V, E)$, $v_i \in V_l, V_l$ represents the collection of microservice components (nodes); $e_{i,j} \in E_l$, that is, the directed edge $v_i \to v_j$, represents the dependency call relationship of the microservice node.

Definition 2: FCDG, labeled as $G_f = (V, E), v_i \in V_f, V_f$ representation of microservice relevance node set, $e_{i,j} \in E_f$, that is, the directed edge $v_i \to v_j$, represents the influence re lationship of microservice relevance nodes.

Definition 3: MFPG, labeled as $G = (V, E), V = V_l \cup V_f$, V represents MFPG fault node, $E = E_l \cup E_f, e_{i,j}$ denotes a directed edge $v_i \to v_j$, MDPG is composed of LCGT and FCDG.

Figure 4 describes the construction method and steps of MFPG: (1) The monitoring tools are used to collect the dependent calls between microservice nodes and various KPI data. When the microservice is abnormal, the data are called according to the abnormal request of the microservice to construct LCGT. (2) Based on the historical fault data of microservice, the fault events of microservice are extracted, and construct FCDG. (3) Combining LCGT and FCDG, the MFPG construction method is designed.

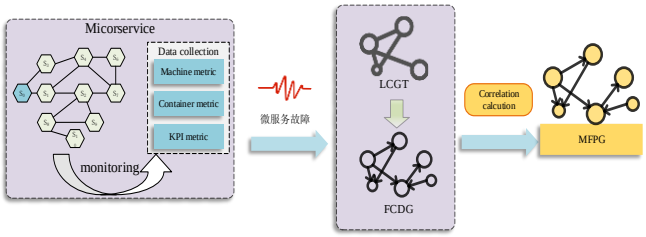

Figure 4 Construction process of MFPG

1) *Call link diagram of end-to-end tracking*

In the field of microservices, the current methods to describe the call relationship of microservice dependency include : manually constructing dependency graph, constructing dependency graph by monitoring port data through system tools, and constructing dependency graph by reasoning relationship. At present, there are some mature service discovery and link tracking mechanisms in microservice software systems. Firstly, this paper designs LCGT construction algorithm for microservice dependent call fault propagation. In order to construct LCGT, the Zipkin link tracking tool commonly used in the microservice system is used to construct the microservice dependency. It has the characteristics of light weight, flexibility and strong scalability, and can track each request call path. In the microservice system, the link tracking service can be completed only by simply configuring the dependency environment and modifying the configuration file. Zipkin ' s trace request call data mainly includes the following properties :

- traceId: A request call of a microservice requires multiple microservice nodes to cooperate. traceId represents the ID of a request call of a microservice, including multiple spanId.
- serviceName: Microservice node name.
- osName: The physical machine or virtual machine where the microservice node is located.
- spanId: spanId represents the ID of a serviceName call.
- parentId: Represents the parent microservice node ID of spanId.
- timestamp: represents the time required for spanId at the microservice node, namely the difference between the call end time and the call start time (endTime-startTime).

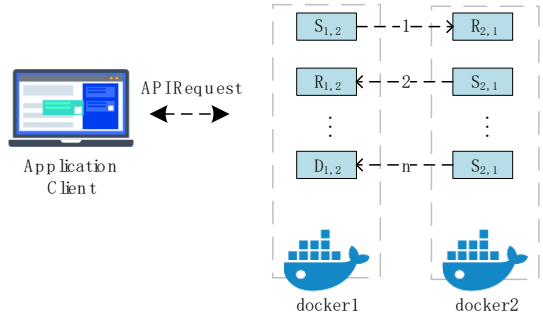

Figure 5 Multiple calls of request execution

Zipkin can track the call information between each micro service component. In a micro service fault event, there are multiple calls between micro service components (such as multiple query databases). As shown in figure 5, docker1 calls docker2 many times, such as $\{(S_{1,2} \to R_{2,1}), \dots, (S_{2,1} \to D_{1,2})\}$, but this paper studies the fault root location of microservices, the purpose is to determine the microservice root components and machine KPI, do not need to pay attention to the interaction details between them, only need to record the docker1 send request time and docker2 return request time in a request, that is to calculate the $S_{1,2}$ send time and $D_{1,2}$ receive time difference. Therefore, the link call data of service request cannot be directly used to generate LCGT. This paper will construct LCGT through the following steps.

Firstly, the microservice component is taken as the node in the figure (microservice node), and the traceId of the request call record is processed. In figure 5, docker1 and docker2 have multiple spanId. After sorting their time, the time difference between the first spanId and the last spanId is used as the response time of docker2, namely Timestamp = (endTime-startTime). Then iteratively compute other nodes to complete the construction of end-to-LCGT. As shown in figure 6, the request path for figure 5 is constructed to generate LCGT through the above steps. Among them, Docker1→Docker2 represents the microservice Docker1 calls Docker2, and Timestamp represents the call response time.

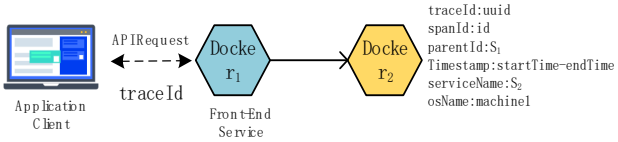

Figure 6. Construct the LCGT process

2) Microservice failure correlation graph

In the historical failure data of microservices, potential failure associations are usually implied. The work content of construct FCDG is divided into two parts: (1) Analyze the fault correlation in microservices from the perspective of fault correlation, and construct a fault correlation graph; (2) Use causal analysis methods to infer the causal relationship between microservice nodes v_i and v_j, $v_i \to v_j$ indicates that the faulty node v_i affects v_j.

1) Correlation analysis of microservice failure

Microservice correlation failure is actually mining the correlation relationship between potential microservice failure nodes. First, relevant symbols and definitions are given to describe the algorithm design process.

Definition 4, Microservice failure event: In the microservice failure, the failure of the microservice from t_{start} to t_{end} is eliminated or the number of microservice failure nodes no longer increases, which is expressed as a microservice failure event f_i, where $f_i = \{s_1, s_2, \dots, s_n,\}$, $1 \le i \le n$, s_i represents the faulty node (such as microservice component, host, container, etc.) in the microservice failure event f_i, n represents the number of microservice nodes, f_i represents the microservice failure time $\Delta t = t_{end} - t_{start}$ involved microservice nodes.

Definition 5, Microservice failure collection: Represents the set of historical fault events of the microservice, expressed as $M^{fault} = \{f_1, f_2, \dots, f_n,\}$, $f_i \in M^{fault}$, f_i is the different microservice fault events in the historical fault of the microservice.

Definition 6, Frequent Microservice Failure Association Set: Given the original microservice failure event set $M^{fault} = \{f_1, f_2, \dots, f_n,\}$, $f_i \in M^{fault}$, there will be no less than the microservice failure event with the given support and confidence f or its subset events are classified as frequent microservice fault correlation set $M^{frequent} = \{f_1, f_2, \dots, f_n,\}$, $f_i \in M^{frequent}$.

When a microservice failure occurs, the two microservice nodes s_i and s_j that do not directly depend on the call in the microservice failure event f_i have a fault relationship. When s_i fails, s_j will also fail immediately, using the form $s_i \to s_j$ or $s_i \leftarrow s_j$ indicates their failure propagation direction, that is, the association rules in the microservice failure event.

In the frequent microservice failure association set $M^{frequent}$, it is assumed that f_i and f_j are two microservice failure events belonging to $M^{frequent}$, and the microservice failure association rules are determined according to support and confidence. Support is used to calculate the frequency of failure events of f_1 or a subset thereof (that is, the frequency or probability of occurrence in $M^{frequent}$); confidence is used to calculate the ratio of the number of common occurrences of failure events f_i and f_j to the number of failure events of f_i or Probability, formulas (1) and (2) represent the support (s) and confidence (c) of calculating $M^{frequent} = \{f_1, f_2, \dots, f_n,\}$.

$$s(f_i) = \frac{\sigma(f_i)}{Mfault} \quad (1)$$

$$c(f_i, f_i) = \frac{\sigma(f_i \cup f_i)}{\nabla(f_i)} \quad (2)$$

Among them, $s(f_i)$ represents the frequency of calculating the microservice failure event f_i in the frequent microservice failure event set $M^{frequent}$, $\nabla(f_i)$ represents the set $f_i \in \nabla(f_i)$ containing the failure time f_i, c (f_i, f_i) Represents the probability that microservice failure events f_i and f_j occur together in $\nabla(f_i)$ (that is, when f_i failure occurs, how likely is the probability of f_j to occur), the greater the value of confidence c, the greater the probability of f_i and f_j The more relevant.

Aiming at the problem of microservice failure relevance, Algorithm 1 provides a pseudo-code algorithm for mining microservice failure relevance. The main steps are as follows:

(1) Enter the set of microservice fault events $M^{fault} = \{f_1, f_2, \dots, f_m, \}$, set the support threshold δ and confidence ε, and first obtain the M^{fault} mid-micro service node as an independent candidate node item set $C_1 = \{s_1, s_2, \dots, s_n\}$, whose elements are candidate node items of length 1 $c = \{s_i\}$, and then calculate each candidate node item according to the support δ, and classify the candidate node items that are not less than δ It is a frequent 1-item set $L_1 = \{s_1, s_2, \dots, s_k\}$.

(2)Combine the frequent item elements of the above L_1 into $C_k^2 = \frac{k!}{2!\cdot(k-2)!}$ candidate node item set with an element length of $C_2 = \{(s_1, s_2), (s_1, s_3), \dots, (s_m, s_n)\}$, and then calculate the support of each candidate node item in the item set C_2, and record the candidate node item with support not less than δ as frequent 2 item set $L_2 = \{(s_1, s_2), (s_1, s_3), \dots, (s_i, s_j)\}$, then the candidate node item in L_2 will be used as the next frequent item element input.

(3) Repeat step (2) until the next level of frequent k items L_k cannot be generated, and finally return all frequent itemsets $\mathbf{L} = \{L_1, L_2, \dots, L_k\}$ not less than the support δ, remove the length from L The frequent element items of 1, because there is only one microservice failure node that does not contain association rules.

(4) In the frequent item set $\mathbf{L} = \{L_1, L_2, \dots, L_t\}$, $L_1 = \{s_1, s_2\}$ indicates that the probability of the two nodes s_1 and s_2 co-occurring in the historical failure event has reached the support index, $s_1 \rightarrow s_2$ or $s_2 \leftarrow s_1$ indicates that the failure of s_1 is relatively large The probability of causing the failure of s_2 or vice versa, this is an association rule that needs to be further mined, and the association relationship between the microservice nodes of L_i can be calculated through the confidence level (c). For this reason, the frequent node items of the frequent itemset L are calculated one by one. First, input the frequent 2 item set $L_2 = \{(s_1, s_2), (s_1, s_3), \dots, (s_i, s_j)\}$, and calculate $c(s_1, s_2)$,$c(s_2, s_1)$,…$c(s_j, s_i)$, and then classify the element group that is not less than the confidence ε into association rule items, and the remaining frequent item set L is iteratively calculated in turn, and finally all the association rule items that meet the conditions are expressed as frequent microservice failures Associative set $M^{frequent} = \{f_1, f_2, \dots, f_n, \}$.

Algorithm 1 Mining Algorithm of Microservice Node Failure Association Set

Input Collection of frequent microservice failure events $M^{fault} = \{f_1, f_2, \dots, f_n, \}$, Support threshold δ, Confidenceε

Output Microservice node failure association collection $\mathbf{L} = \{S_1, S_2, \dots, S_k\}$

```
C_1 = ExtractCandidate(M^fault, 0);
L_1 = FrequentItemsets(C_1, δ);
if L_1 ≠ ∅;
  while( L_k ≠ ∅ ) do
    C_{k+1} = ExtractCandidate(M^fault, L_k);
    L_{k+1} = FrequentItemsets(C_{k+1}, δ);
    S_{k+1} = ExtractFaultCorrelationSet(L_{k+1}, ε);
    k=k+1;
  end while
end if
L = {S_1, S_2, ..., S_k};
```

According to Algorithm 1, a collection of microservice nodes with fault association rules and fault association is obtained. When a microservice failure occurs, the propagation path and impact range of the failure can be described according to the associated set of microservice node failures. However, the fault association rule only characterizes that when the microservice node v_i fails, it is very likely that the propagation will cause the node v_j to also fail. However, the propagation direction of the microservice failure cannot be determined in the event of a failure. Therefore, the next step is to determine the fault propagation direction of the microservice node, calculate the causal influence relationship between the faulty nodes through the causal relationship analysis method, infer the fault propagation direction, and construct the FCDG.

The microservice KPIs monitored and collected in this paper belong to time series data. Among the causality analysis methods, Granger's method[19] is widely used in time series data. This method can be used to calculate the causality between microservice nodes. The formal definition of Granger causality based on time series is as follows: Given two time series $X_n = \{x_1, x_2, \dots, x_n\}$ and $Y_n = \{y_1, y_2, \dots, y_n\}$, if the time series X_{n-t} can Predicting the value of x_t at the time point indicates that the time series X_{n-t} and x_t have a causal relationship, and X_{n-t} is a factor that affects x_t, such as formula (3), which is limited to the inference of its own causality.

To infer the causality between x and y, it is necessary to extend equation (3) to two-dimensional Granger causality analysis. Given the time series X_{n-t} and Y_{n-t}, if the value of x_t is predicted based on the time series X_{n-t} and Y_{n-t}, the value of x_t is more accurate (smaller error) than the prediction of x_t based only on X_{n-t}, It means that x and y have a Grange causal relationship, that is, y is a factor that affects x. The calculation formula is shown in (4).

$$x_t = \beta_0 + \sum_{k=1}^{p} \beta_k x_{t-k} + \varepsilon_x \tag{3}$$

$$x_t = \theta_0 + \sum_{k=1}^{p} \beta_k x_{t-k} + \sum_{n=1}^{q} \theta_k y_{t-n} + \varepsilon_{y,x} \tag{4}$$

Among them, θ and β are coefficients, ε is the prediction error (predicted value-true value), p and q are the length of the lag time series relative to the current time t.

Based on formulas (3) and (4), the Granger causality between y and x can be inferred. If $\varepsilon_x > \varepsilon_{y,x}$, it means that y has a better accuracy improvement for the prediction result of x, that is Y is a cause of x, otherwise it means that y has no causal relationship with x.

$$F_{y \rightarrow x} = \ln \frac{\varepsilon_{y,x}}{\varepsilon_x} \tag{5}$$

According to the above expression, the Granger causality between time series data X and Y is calculated and inferred using formula (6).

$$F_{y \rightarrow x} = \begin{cases} 0, & \text{no } Granger \\ F_{y \rightarrow x} > 0 \end{cases} \tag{6}$$

According to the above formula, the Granger causality analysis method is applied to the microservice fault related nodes to calculate the fault propagation direction.

The main steps of Grange causality analysis for microservice failure related nodes are as follows:

(1) Obtain the microservice fault correlation set based on Algorithm 1, extract a group of microservice nodes (v_i, v_j) with fault correlation as input, and obtain the KPI time series data of the microservice nodes, (v_i, v_j) corresponds to The two sets of KPI time series are $X_n = \{x_1, x_2, \dots, x_n\}$ and $Y_n = \{y_1, y_2, \dots, y_n\}$.

(2) Use formulas (4) and (6) to perform Granger causality analysis on each microservice node (v_i, v_j), and calculate and infer the fault propagation direction $v_j \to v_i$ or $v_i \to v_j$.

(3) Repeat steps 1 and 2 to calculate the causality of all nodes, and finally generate FCDG.

In summary, the end-to-end tracking call link graph LCGT and the microservice fault correlation directed graph FCDG are constructed, and the construction of two fault propagation graphs is completed. When a microservice failure occurs, Algorithm 2 will build MFPG based on LCGT and FCDG. The main construction process is described as follows:

Algorithm 2. MFPG construction algorithm based on fault correlation

Input Call link diagram for end-to-end tracking $G_l = (V_l, E_l)$、Microservice fault correlation directed graph $G_f = (V_f, E_f)$

Output MFPG: $G = (V, E)$

```
01 Initialize according to definition 3.8 G = (V, E), |V| = |V_l| + |V_f|
02 while (|V_l| ≠ ∅) do
03   V ← V_l ;
04   E ← E_l ;
05 end while
06 while (|V_f| ≠ ∅) do
07   if V_f ¬∈ V do
08     V ← V_f ;
09     E ← E_f ;
10   end if
11 end while
```

Algorithm 2 describes the process of constructing a microservice fault propagation relationship graph. Input two fault graphs LCGT and FCDG, and add the microservice nodes and directed edges of LCGT and FCDG to graph G successively. Among them, the seventh line needs to remove the same nodes that may exist in LCGT and FCDG, and then iteratively add nodes and directed edges to the graph to complete the construction of MFPG. Algorithm 2 adds the microservice node and directed edge to the MFPG through two loop statements, and the time complexity is $O(|V_l| + |V_f|)$, $|V_l| + |V_f| \in R$.

3) Microservice failure root cause location

Aiming at the dynamic displacement characteristics of micro service failure KPI, this paper designs an anomaly scoring algorithm based on correlation analysis to measure the anomaly degree of each node in MFPG, and introduces the cross correlation function[20] to calculate whether the two groups of time series data have lag or noise interference. As shown in figure 7, the cross-correlation function is used to calculate the cross-correlation of the two groups of KPIs, so as to obtain the maximum cross-correlation value of the two groups of KPIs at this point. Therefore, the degree of displacement between KPIs can be calculated by using cross-correlation function, and then the correlation coefficient of KPI sequence after displacement can be calculated.

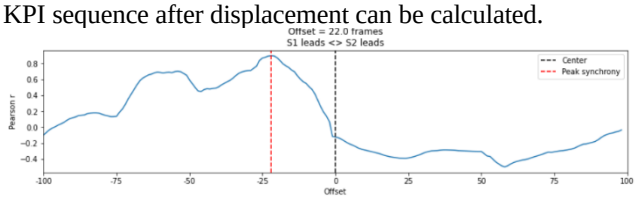

Figure 7 KPI cross-correlation value

this paper designs a correlation analysis-based anomaly scoring algorithm c-SBD (correlation SBD) based on SBD[21] to effectively solve the displacement problem of microservice KPIs and measure the degree of microservice failures.

Design of the exception scoring algorithm: Given two sets of microservice KPI time series data $X = (x_1, x_2, \dots, x_m)$ and $Y = (y_1, y_2, \dots, y_m)$, the principle of mutual correlation calculation is to keep the time series Y static and slide X to the right in steps of 1 unit length and calculate the the inner product of each value. Where, s denotes the displacement length, $s \geq 0$ means move X to the right, $s < 0$ means move X to the left, then the displaced X is denoted as (7).

$$\mathrm{X} = \begin{cases} (0, \dots, 0, x_1, x_2, \dots, x_{m-s}), & s \geq 0 \\ (x_{1-s}, \dots, x_{m-1}, x_m, 0, \dots, 0), & s < 0 \end{cases} \tag{7}$$

where the number of zeros is |s|, the constraint is $w = m/2, s \in [-w, w]$, and w denotes the degree of displacement of the KPI data; if the displacement distance is too large, the correlation of microservice KPIs cannot be analyzed effectively. The intercorrelation calculation results in a set of temporal data with a length of 2m-1 , expressed as: $\mathrm{CC\_}w\ (X, Y) = (c\_1, c\_2, \dots, c\_m)$, as shown in equation (8).

$$\mathrm{CC}_w(X, Y) = \begin{cases} \sum_{i=1}^{m-s} x_i \cdot y_{s+i}, & s \geq 0 \\ \sum_{i=1}^{m+s} x_{i-s} \cdot y_i, & s < 0 \end{cases} \tag{8}$$

Because of the different types of microservice KPIs, they are normalized in order to calculate the relevance, and then normalized as shown in equation (9).

$$\mathrm{NCC(X, Y)} = max\left(\frac{\mathrm{CC}_w(X, Y)}{\|X\|_2 \cdot \|Y\|_2}\right) \tag{9}$$

Finally, the formula for calculating the anomaly score is shown in (10).

$$\mathrm{c - SBD(X, Y)} = 1 - \mathrm{NCC(X, Y)} \tag{10}$$

where $NCC(X, Y) \in [-1, 1]$, then c-SBD takes values in the range [0,2], with 0 indicating perfect correlation and the larger the value the lower the correlation.

The above work completes the key design of the exception scoring algorithm, and the next step will be to use Equation (10) to measure the degree of exception of the microservice nodes, which is calculated in two cases when measuring the degree of exception of the microservice nodes.

(1) MFPG is mainly composed of LCGT and FCDG fault propagation diagrams. When calculating and evaluating the abnormal degree of microservice nodes in LCGT, the response time KPI of microservice is used as the calculation parameter of abnormal score. Let $S(R(v_i), R(avg))$ denote

the exception score of microservice node v_i, $R(avg)$ denotes the average response time KPI data of microservice system, and $R(v_i)$ denotes the response time KPI data of node v_i, and calculate the exception score of microservice node v_i according to equation (10), then the exception score is shown in equation (11).

$$\mathrm{S}\big(R(v_i), R(avg)\big) = 1 - max\left(\frac{\mathrm{CC}_w\big(R(avg), R(v_i)\big)}{\|R(avg)\|_2 \cdot \|R(v_i)\|_2}\right) \quad (11)$$

(2) When evaluating the degree of node exceptions in FCDG, R(avg) of the microservice system and machine KPI data of FCDG node $v_i$are used as the calculation parameters of the exception score. Denote $M_i = \{cpu, memory, tcp, \dots\}$for the node machine KPI of FCDG, and $M_i(x)$for the class x machine KPI timing data of node v_i. The anomaly score of microservice node v_i is calculated according to Equation (10), and the maximum value of machine KPI correlation coefficient is used as the exception score of node v_i, and the exception score is shown in Equation (12).

$$\mathrm{S}\big(R(v_i), R(avg)\big) = max\left(1 - max\left(\frac{\mathrm{CC}_w\big(R(avg), \mathrm{M}_i(x)\big)}{\|R(avg)\|_2 \cdot \|\mathrm{M}_i(x)\|_2}\right)\right) \quad (12)$$

Finally, the exception score metric defined by Eqs. (11) and (12) is extended to MFPG to calculate the exception score of microservice as shown in Eq. (13).

$$\mathrm{S}\big(R(v_i), R(avg)\big) = \begin{cases} 1 - max\left(\frac{\mathrm{CC}_w\big(R(avg), R(v_i)\big)}{\|R(avg)\|_2 \cdot \|R(v_i)\|_2}\right), v_i \in DEG \\ max\left(1 - \max\left(\frac{\mathrm{CC}_w\big(R(avg), \mathrm{M}_i(x)\big)}{\|R(avg)\|_2 \cdot \|\mathrm{M}_i(x)\|_2}\right)\right), v_i \in FCG \end{cases} \quad (13)$$

At this point, this section completes the design of the microservice exception scoring algorithm, and the next step is to infer the root cause of microservice failures based on the degree of exceptions of microservice nodes.

Random wandering optimization algorithm: In the microservice fault root cause location phase, the microservice KPI data is collected through subsection 3.1, and the exception score of each node in MFPG is calculated according to the exception scoring algorithm, and then the access probability P(i) of each node is obtained, as in equation (14).

$$P(i) = \frac{\mathrm{S}\big(R(v_i), R(avg)\big)}{\sum_{i=1}^{n} \mathrm{S}\big(R(v_i), R(avg)\big)} \quad (14)$$

where n denotes the number of nodes of the MFPG.

Random wandering is a probabilistic search algorithm, for which the formal symbolic representation $G = (V, E)$ of the microservice fault propagation relation graph is first defined, and its transfer probability matrix P, $e_{k,i} \in E$ denotes the node fault propagation direction $v_k \to v_i$, then the probability of transferring access to node v_i from node v_k is $[P]_{k,i} = p_{k,i}$.

In order to enhance the global and flexibility of the search algorithm and avoid getting stuck in the low exception region during the search process and unable to get out, three search access directions are designed: forward access, backward access, and stay access, corresponding to the access probabilities of $p_i^{forward}$, p_i^{back}, and p_i^{self}, then the main steps of the algorithm are as follows.

(1) Forward access: Assuming that node v_a is the starting node of the search algorithm and searches along the topological direction of the MFPG, v_a denotes the current node, v_i denotes the next visited node, $e_{a,i} \in E$, and v_i is a child of v_a, the probability of visiting node $v_i$is as in equation (15), and $p_{a,i}^{forward}$ denotes the probability of transferring from node v_a to visit node v_i.

$$p_{a,i}^{forward} = P(i) \quad (15)$$

Where v_i is a child of node v_a, and the child with a larger exception score will have a greater probability of access.

(2) Backward access: The random walk algorithm may enter the low correlation region in the search access process, so it cannot obtain the global optimal results. In order to avoid this situation, backward access is designed. When a microservice node v_j with low exception score is visited, and the exception scores of v_j and all its children are less than a given value σ (e.g., set to the mean value of the exception score), the search algorithm will visit along the opposite direction of the directed edge $e_{i,j} \in$ E and return to its parent node $v_i$with probability $p_{j,i}^{back}$ from node v_j, and then reselect a child node to be visited with the backward visit probability as in equation (16).

$$p_{j,i}^{back} = \rho P(i) \quad (16)$$

where $\rho \in [0,1]$ is a custom backward access parameter, and the strength of backward traversal is changed by setting ρ. If ρ takes a small value, the random wandering algorithm will continue the search more strictly in the direction of the topology graph; if ρ takes a larger value, the random wandering algorithm will be more flexible for global access search; MFPG will take a larger value of ρ if most of the nodes belong to the dependency invocation relationship, and vice versa, the smaller the value of ρ.

(3) Stay visit: microservice nodes with higher exception scores are more likely to be associated with the root cause of the failure; if the current visiting node has a higher exception score than all its neighbors, and the exception scores of the neighboring nodes are all less than a given threshold σ, the current node is visited again with p_i^{self}, increasing the number of visits to $v_i$with visit probability as in equation (17).

$$p_i^{self} = \max\left(0, \frac{\mathrm{S}\big(R(v_i), R(avg)\big) - max_{j:e_{j,k} \in E}\big(\mathrm{S}\big(R(v_i), R(avg)\big)\big)}{\sum_{i=1}^{n} \mathrm{S}\big(R(v_i), R(avg)\big)}\right) \quad (17)$$

where $max_{j:e_{j,k} \in E}\big(S(v_k, cSBD)\big)$notes the neighbor node with the highest exception score.

So far, the random wandering based fault rooting in optimization algorithm is completed. The traditional random walk algorithm is optimized by designing three search directions: forward visit, backward visit, and stay visit, which enhances the global and flexible search scope of the algorithm.

C. Design of the MFRL-CA algorithm

Our MFRL-CA Algorithm 3 is designed for microservice fault root cause location analysis. First, the transfer probability matrix P of MFPG is calculated according to the exception scoring algorithm, then the fault nodes visiting MFPG are flexibly searched by the optimized random walk algorithm during the fault root cause location, and finally the number of visits to the fault nodes is counted and the Top-k fault root causes are ranked by iterative traversal search. The design process of MFRL-CA is given in Algorithm 3.

Algorithm 3: Randomized wandering root cause inference algorithm ba sed on MFPG.

Input: $G = (V, E)$, n
Output: Root rank $\mathbf{R}[n]$
01. Initialize the transfer probability matrix *P*, the root factor arr ay R[n]
02. $v_a \leftarrow random(V)$;
03. **while($|V| \neq 0$) do**
04. $[\mathbf{P}]_{(a,i)} \leftarrow p_{a,i}^{forward}$;
05. $[\mathbf{P}]_{(j,i)} \leftarrow p_{j,i}^{back}$;
06. $[\mathbf{P}]_{(i,i)} \leftarrow p_{i}^{self}$;
07. **end while**
08. **while(n $\neq$ 0) do**
09. $v_i \leftarrow v \in O_a \cup I_a \cup \{v_a\}$;
010. $\mathbf{R}[s] \leftarrow \mathbf{R}[s] + 1$;
011. n = n − 1;
012. **end while**
013. $\mathbf{R}[n] \leftarrow Sort(\mathbf{R}[n])$;

Based on MFPG, Algorithm 3 calculates the anomaly score of each microservice node by the exception scoring algorithm and generates a transfer probability matrix of $G = (V, E)$, then designs a random wandering algorithm with three search directions: forward, backward and stay-visit, which avoids the algorithm getting stuck in a local area and unable to get out of the trap, thus optimizing the global dynamic search capability of the root cause location algorithm. Finally, the MFPG is accessed by multiple iterations of search, and the number of microservice node visits is counted to rank the top Top-k fault root causes. The time complexity of Algorithm 3 to calculate the anomaly score of MFPG nodes is $O(n^2)$, and n is the number of microservice nodes; in the random wandering search access phase, if the number of user-defined search accesses is m, the time complexity is $O(m)$, and the total time complexity is $O(n^2 + m)$.

V. EXPERIMENT

In this section, we evaluate MFRL-CA based on the our microservice system based shopping platform.

A. Experimental Setup

The template is designed so that author affiliations are not repeated each time for multiple authors of the same affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization). This template was designed for two affiliations.

1) Baseline Algorithms

Random Selection (RS): Random selection method, according to the type of fault randomly selected fault micro service components for investigation.

Sudden Change (SC): By checking whether there is a mutation in the data of the two adjacent time windows, if there is a mutation in the data of one of the time windows, the average ratio of the data of the two time windows is calculated as a measure of the root cause score of the node, and the possibility of the root cause is judged according to the score.

Timing Behavior Anomaly Correlation (TBAC)[22]: Through the service dependence call graph technology anomaly correlation, and infer the root cause of the fault.

2) Evaluation Metric

In the field of microservice fault root cause location, literature [12] earlier proposed two commonly used evaluation indicators, the highest K accuracy rate ($AP@K$) and Mean Avaerage Precision (MAP). The fault root cause location algorithm usually outputs Top-k root cause indicators. These two evaluation indicators can quantify the effectiveness of the algorithm.

For abnormal event A, |A| means abnormal set, $I_a(i)$ indicates whether microservice node v_i is the true root cause of failure event A, $I_a(i) = 1$ means true, $I_a(i) = 0$ means False; $\varphi_a(i)$ indicates that in the output ranking of the algorithm, the fault root cause node v_i is ranked. For example, when Top-k=3, it means that the algorithm will output 3 possible root cause indicators, where $\varphi_a(1)$ represents the microservice node v_1 ranks first in the output root cause.

The highest K accuracy rate ($AP@K$)[12]: It indicates the probability that the top-k root cause indicators of the root cause in the output ranking of the root cause location algorithm contain the true root cause of the fault. The calculation formula is as (18).

$$AP@K = \frac{1}{|A|}\sum_{a\in A}\frac{\sum_{i:\varphi_a(i)\le K} I_a(i)}{min(K, \sum_i I_a(i))} \tag{18}$$

Average accuracy rate (MAP)[12]: Measure the overall accuracy rate of the algorithm, where |V| represents the number of nodes in the microservice fault propagation relationship graph $G = (V, E)$, and the calculation formula is as (19).

$$MAP = \frac{1}{|V|}\sum_{1\le k\le |V|} AP@K \tag{19}$$

Average False Positives (AFP): $F_a(i)$ represents the number of faulty nodes that have been checked before finding the true root cause node v_i, then $F_a(i)$ calculation formula is as (20), AFP calculation formula is as (21).

$$F_a(i) = \begin{cases} \sum_{j=1}^{\varphi_a(i)} \left(1 - I_a(i)\right), & I_a(i) = 1 \\ 0, & I_a(i) = 0 \end{cases} \tag{20}$$

$$AFP = \frac{1}{|A|}\sum_{1\le k\le |V|}\frac{\sum_{1\le i\le |V|} F_a(i)}{\sum_{1\le i\le |V|} I_a(i)} \tag{21}$$

3) Datasets and Implementation

This section divides the types of faults into three categories to evaluate the effect of the algorithm, including resource overhead faults (CPU HOG: injecting high CPU load through faults to increase CPU utilization to 95%) and network latency faults (Latency: Increase the network delay

between microservice components, the delay time is 10ms-50ms), microservice software failure (Throughput: increase the amount of request concurrency through script programs). Therefore, in order to evaluate the performance of the algorithm, not only the two fault propagation conditions must be considered, but also the mixed faults.

First, consider the failures caused by the dependence of microservice on the calling relationship, and simulate the failure of a microservice component in the microservice system and spread to other normal components through LCGT. As shown in figure 8, the fault injection tool causes high CPU utilization and high latency of the microservice component Payment, and affects other service components through LCGT. For example, the response time of the microservice components had abnormal changes such as Payment, Orders, Catalogue, etc.

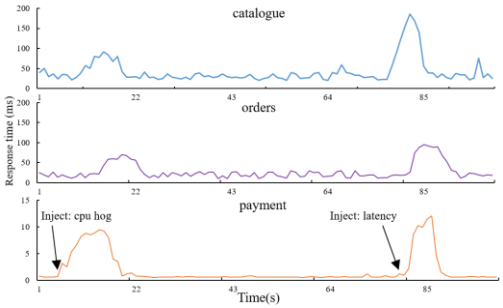

Figure 8. Reliance on the call relationship leads to failure conditions

Second, simulate the relationship between microservice failures and inject failures into other service components of the microservice server. By increasing the CPU utilization rate of the Stress container, it interferes with the performance of other microservice components of the same server, and finally causes the microservice system to fail. Happening.

Third, In fault injection, three types of faults are randomly injected into the microservice components with call dependence by simulating mixed faults.

B. Simulation Experiment and Analysis

Firstly, the three types of fault data sets, Throughput, CPU HOG, and Latency, are tested to verify and evaluate the accuracy of the algorithm and MAP.

As shown in figure 9, this paper evaluates the effect of the algorithm under different k conditions. The MFRL-CA algorithm is better than other methods under different k values. Compared with the traditional random selection algorithm (SC) And the mutation detection algorithm (RS) accuracy rate has been greatly improved; the TBAC algorithm and the algorithm in this paper are different in the modeling of microservice failure propagation and diffusion relationship. The former only considers the dependency call relationship of microservice, while the algorithm in this paper MFPG is constructed by combining LCGT and FCDG, and the flexibility and globality of the root cause search algorithm is optimized, and the accuracy rate is about 11% higher than that of the TBAC algorithm. Among them, the accuracy of the root cause location algorithm increases with the increase of k. This is because when the value of k is larger, the Top-k root cause set will cover more fault root cause indicators, and the output true fault root cause Because the probability of failure is greater. If the value of k is much larger than the number of true fault root causes of the fault event sample, the output result will output more non-true fault root cause indicators, which will lead to more false positive data.

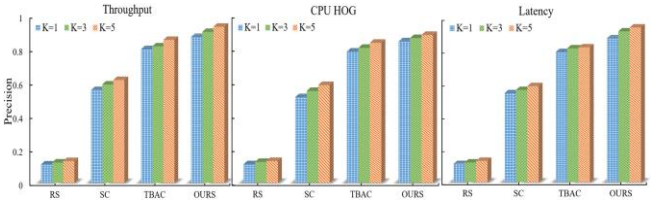

Figure 9. MFRL accuracy evaluation under different fault types

As shown in figure 10, the AP@K and MAP of the MFRL-CA algorithm in the experimental verification are comprehensively evaluated. The average accuracy MAP of MFRL-CA is 89.75%, which is better than other algorithms. The MFRL-CA can quickly, accurately and automatically locate the root cause of the fault by collecting the abnormal KPI data of the faulty node, constructing the MFPG, anomaly scoring, and locating the root cause of the fault after a microservice fault is discovered, achieving a minute-level fault root. Because of positioning. Compared with the traditional operation and maintenance methods RS and SC, the timeliness, accuracy, and operation and maintenance efficiency are greatly improved.

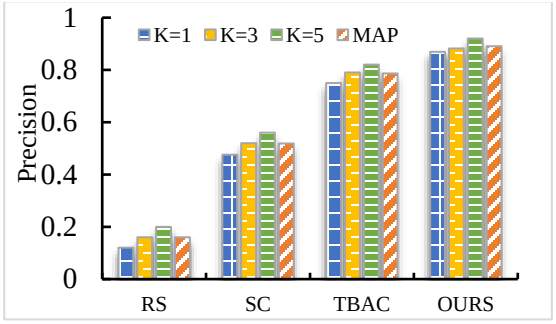

Figure 10. MFRL-CA method Precision and MAP

C. Reliability assessment

In order to evaluate the reliability of the algorithm, microservice container instances of different scales were deployed in the experiment. In the experiment, multiple Sock Shop microservice container instances were generated through the Docker image, and other service container instances were also added, and the microservice scale was set to 20, 30, 40, and 50.

Figure 11 shows the average accuracy MAP of each fault root cause location algorithm in different microservice scales. It can be seen that this article still has a high accuracy rate in different microservice cluster scale environments, and they are all higher than the traditional manual operation and maintenance algorithms. Among them, in the scenario where the microservice scale is 50, MAP is 12.7% higher than the TBAC algorithm, which has better stability and accuracy. The 4 types of root cause fault location algorithms have decreased as the scale of microservice increases, but the MAP decline trend of the algorithm in this paper is relatively slow. This is

because the algorithm in this paper optimizes the fault search in the process of locating the root cause of microservice faults. The performance of the algorithm is designed with three search and access directions. Compared with other algorithms, it has better flexibility and search globality, avoids falling into the local optimal solution, and improves the reliability of the algorithm. The method in this paper is better than other methods in different service scale scenarios, and will not suffer a large performance degradation as the service scale increases. It can be adapted to microservice of different scales and has better reliability and accuracy.

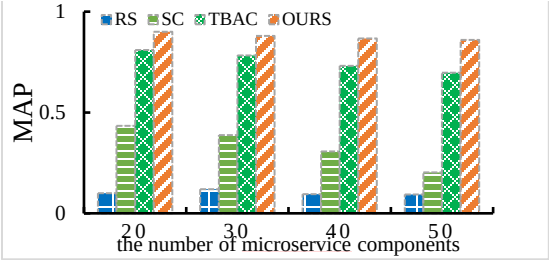

Figure 11. MAP of root cause location algorithm under different microservice scales

As shown in figure 12, comparing the AFP index of the algorithm, the algorithm in this paper is significantly lower than other algorithms under different micro service scales. According to the statistics of AFP index of this algorithm, we only need to check 0.45 microservice fault nodes one by one to find the real root cause index, realize the automatic location of fault root cause, and reduce the time of microservice fault location.

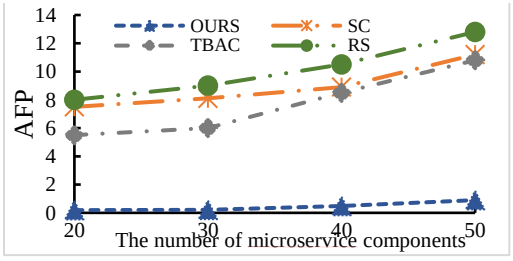

Figure 12. AFP under different microservice scales

D. *Evaluation of the microservice fault propagation graph*

It verifies the effectiveness and necessity of the microservice fault propagation relationship graph MFPG for root cause location algorithms, and compares and evaluates it with three commonly used fault root cause location methods.

Correlation analysis method (CA): In microservice failures, traditional operation and maintenance engineers calculate the Pearsion correlation by comparing the faulty node with abnormal indicators or fault types. High correlation may be the root cause of the fault. Correlation analysis between the KPI of the faulty node and the KPI of the response time of the microservice.

Anomaly Clustering (AC): clustering microservice components with similar abnormal patterns. The input data of the clustering is the microservice KPI, and the distance calculation formula uses DTW. This method is not based on MFPG.

Random walk algorithm based on dependent call graph (LCGT+RW): Different from the MFRL-CA algorithm of this article, only the LCGT of this article is used as the fault graph, and the root cause of the fault location phase uses the optimized random walk algorithm of this article to search and visit LCGT and list the root causes of the failure.

As shown in figure 13, abscissa is the size of microservice fault data set verified in each test. With the increasing test data, the accuracy of each method tends to a stable value. The algorithm in this paper is superior to other methods, and the accuracy is 10.4% higher than LCGT+RW method. Both CA and AC methods do not rely on the micro service dependency call graph. When the fault data set is small, the accuracy fluctuates greatly. This is because the CA method directly calculates the node correlation as the basis for troubleshooting the root cause of the fault; AC method is to cluster the same microservice components. When the KPI of the faulty microservice component has a high similarity pattern (such as injecting CPU hog), it can not find out the root cause index effectively. LCGT+RW method can not accurately locate the root cause index of micro service associated fault because it can not describe the fault propagation path.

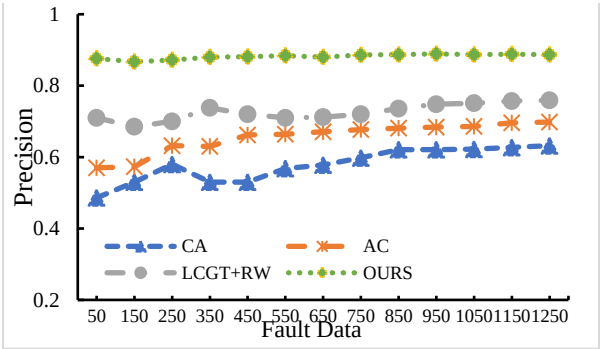

Figure 13 Comparison of accuracy of different fault root cause location methods

According to the review, the MFRL-CA algorithm has a higher accuracy rate than the CA, AC, and LCGT+RW methods, because MFPG characterizes two failure propagation paths. When random walk search is used to access the fault root, not only the correlation coefficient of the node is considered, but also the propagation and diffusion probability of the fault is considered. The correlation coefficient also takes into account the probability of the propagation and spread of the fault, and the area with a higher score will get more visits, which verifies the effectiveness and necessity of MFPG for locating the root cause of the fault.

VI. CONCLUSION

Aiming at the problem that the fault relationship of micro service is complex and it is difficult to locate the root cause of the fault. How to ensure the QoS of microservice, reduce the losses caused by microservice failures, and improve the efficiency of operation and maintenance is a key issue in the field of operation and maintenance. Therefore, this paper studies the relationship between the microservice fault propagation graph and the fault root cause location method,

and proposes a MFRL-CA method, which can quickly and accurately locate the root cause of the fault after the fault occurs, and automatically give the root cause index of the fault to ensure the microservice QoS, reduce the loss caused by microservice failure, and improve the efficiency of operation and maintenance.

ACKNOWLEDGMENTS

This work is supported by the Natural Science Foundation of China (No. 61762008), the National Key Research and Development Project of China (No. 2018YFB1404404), the Major special project of science and technology of Guangxi (No.AA18118047-7), and the Guangxi Natural Science Foundation Project (No. 2017GXNSFAA198141).

REFERENCES

[1] Zhang, H., Li, S., Jia, Z., et al, "Microservice architecture in reality: An industrial inquiry. In 2019 IEEE International Conference on Software Architecture (ICSA) (pp. 51-60). IEEE, doi: 10.1109/ICSA.2019.00014.

[2] 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, doi: 10.1109/TSE.2018.2887384.

[3] Ma S P, Fan C Y, Chuang Y, et al, "Using service dependency graph to analyze and test microservices," 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2018, 2: 81-86, doi: 10.1109/COMPSAC.2018.10207.

[4] Dapper[EB/OL]. https://github.com/DapperLib/Dapper.

[5] Zhou X, Peng X, Xie T, et al, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2019: 683-694, doi:10.1145/3338906.3338961.

[6] Chiba S, "Javassist—a reflection-based programming wizard for Java," Proceedings of OOPSLA'98 Workshop on Reflective Programming in C++ and Java. ACM, 1998, 174: 21.

[7] Zhao X, Rodrigues K, Luo Y, et al, "Log20: Fully automated optimal placement of log printing statements under specified overhead threshold," Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017: 565-581, doi: 10.1145/3132747.3132778.

[8] Zhao X, Rodrigues K, Luo Y, et al, "Non-intrusive performance profiling for entire software stacks based on the flow reconstruction principle," 12th USENIX Symposium on Operating Systems Design and Implementation. USENIX, 2016: 603-618.

[9] Marwede N, Rohr M, van Hoorn A, et al, "Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation," 2009 13th European Conference on Software Maintenance and Reengineering. IEEE, 2009: 47-58, doi: 10.1109/CSMR.2009.15.

[10] Chen P, Qi Y, Zheng P, et al, "CauseInfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems," IEEE INFOCOM 2014-IEEE Conference on Computer Communications. IEEE, 2014: 1887-1895, doi: 10.1109/INFOCOM.2014.6848128.

[11] Zi-Yong W, Tao W, Wen-Bo Z, et al, "Fault Diagnosis for Microservices with Execution Trace Monitoring," Journal of Software, 2017, 28(6): 1435-1454, doi:10.13328/j.cnki.jos.005223.

[12] Kim M, Sumbaly R, Shah S, "Root cause detection in a service-oriented architecture," ACM SIGMETRICS Performance Evaluation Review, 2013, 41(1): 93-104, doi:10.1145/2494232.2465753.

[13] Ma M, Lin W, Pan D, et al, "Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications," 2019 IEEE International Conference on Web Services (ICWS). IEEE, 2019: 60-67, doi:10.1109/ICWS.2019.00022.

[14] Wang P, Xu J, Ma M, et al, "Cloudranger: Root cause identification for cloud native systems," 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). IEEE, 2018: 492-502, doi:10.1109/CCGRID.2018.00076.

[15] Zi-Yong W, Tao W, Ji-Wei X, et al, "A Survey of Fault Detection for Distributed Software Systems with Statistical Monitoring in Cloud Computing," Chinese Journal of Computers, 2017, 40(02): 397-413, doi:10.11897/SP.J.1016.2017.00397.

[16] Ismail H, Riasetiawan M, "CPU and memory performance analysis on dynamic and dedicated resource allocation using XenServer in Data Center environment," 2016 2nd International Conference on Science and Technology-Computer, Yogyakarta, 2016: 17-22, doi:10.1109/ICSTC.2016.7877341.

[17] Lu-lu N, Hong-jie J, "Transient Event Detection Algorithm for Non-intrusive Load Monitoring," Automation of Electric Power Systems, 2011, 35(09): 30-35, doi:CNKI:SUN:DLXT.0.2011-09-007.

[18] Akoglu L , Tong H , D Koutra, "Graph-based Anomaly Detection and Description: A Survey," Data Mining & Knowledge Discovery, 2015, 29(3):626-688, doi:10.1007/s10618-014-0365-y.

[19] Qiu H, Liu Y, Subrahmanya N A, et al, "Granger causality for time-series anomaly detection," 2012 IEEE 12th international conference on data mining. IEEE, 2012: 1074-1079, doi:10.1109/ICDM.2012.73.

[20] Sakurai Y, Papadimitriou S, Faloutsos C, "Braid: Stream mining through group lag correlations," Proceedings of the 2005 ACM SIGMOD international conference on Management of data. ACM, 2005: 599-610, doi:10.1145/1066157.1066226.

[21] Paparrizos J, Gravano L, "k-shape: Efficient and accurate clustering of time series," Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015: 1855-1870, doi:10.1145/2949741.2949758.

[22] Marwede N, Rohr M, van Hoorn A, et al, "Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation." 2009 13th European Conference on Software Maintenance and Reengineering. IEEE, 2009: 47-58, doi:10.1109/CSMR.2009.15.