Received November 19, 2020, accepted December 8, 2020, date of publication December 14, 2020, date of current version December 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3044610

An Anomaly Detection Algorithm for Microservice Architecture Based on Robust Principal Component Analysis

MINGXU JIN[1], AORAN LV[1], YUANPENG ZHU[1], ZIJIANG WEN[2], YUBIN ZHONG[2], ZEXIN ZHAO[1], JIANG WU[1], HEJIE LI[1], HANHENG HE[1], AND FENGYI CHEN[3]
[1]School of Mathematics, South China University of Technology, Guangzhou 510641, China
[2]JOYY Inc., Guangzhou 511442, China
[3]School of Management Administration, Guangdong Industry Polytechnic, Guangzhou 510330, China

Corresponding author: Yuanpeng Zhu (ypzhu@scut.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61802129, and in part by the Natural Science Foundation of Guangdong Province, China, under Grant 2018A030310381.

ABSTRACT Microservice architecture (MSA) is a new software architecture, which divides a large single application and service into dozens of supporting microservices. With the increasingly popularity of MSA, the security issues of MSA get a lot of attention. In this paper, we propose an algorithm for mining causality and the root cause. Our algorithm consists of two parts: invocation chain anomaly analysis based on robust principal component analysis (RPCA) and a single indicator anomaly detection algorithm. The single indicator anomaly detection algorithm is composed of Isolation Forest (IF) algorithm, One-Class Support Vector Machine (SVM) algorithm, Local Outlier Factor (LOF) algorithm, and 3σ principle. For general and network time-consuming anomaly in the process of the MSA, we formulate different anomaly time-consuming detection strategies. We select a batch of sample data and three batches of test data of the 2020 International AIOps Challenge to debug our algorithm. According to the scoring criteria of the competition organizers, our algorithm has an average score of 0.8304 (The full score is 1) in the four batches of data. Our proposed algorithm has higher accuracy than some traditional machine learning algorithms in anomaly detection.

INDEX TERMS Microservice architecture, root cause analysis, anomaly detection.

I. INTRODUCTION

The concept of microservice emerged in 2012. As a way to speed up the development process of web and mobile applications, it has been widely concerned since 2014. MSA is an architecture concept. Its main idea is to decompose a big service into discrete services, to reduce system coupling, and provide more flexible service support. MSA divides a large single application and service into several or even dozens of supporting microservices. It can expand a single component rather than the entire application stack to meet the service level agreement [1], [2].

The MSA makes the responsibilities of each part of the system clearer. Each part is dedicated to providing better services for others, by this strategy, there are still some problems in the MSA [3]–[6]:

- The whole application of MSA is divided into multiple services, so it is difficult to locate the failure nodes.
- In the MSA, a service failure may produce an avalanche effect, resulting in the whole system failure.

Therefore, the fault diagnosis and root cause analysis of MSA is particularly important. Currently, most studies of root cause analysis (RCA) rely mainly on monitoring data, including log data, service dependency data, and invocation chain data [7]–[11].

In recent years, there are more and more RCA algorithms and anomaly detection algorithms depending on graphs. The researches of some MSA and cloud applications are very representative [9]–[21].

Ma *et al.* presented iSQUAD, a framework for iSQ root cause diagnoses, to deal with the problem of intermittent slow

The associate editor coordinating the review of this manuscript and approving it for publication was Bohui Wang.

queries (iSQs) in real-world cloud databases [9]. Nie *et al.* proposed an automatic diagnosis system based on causality graph to help system operators find the root causes [10]. Xu *et al.* presented a novel approach to API performance monitoring, which recognizes performance problems by response time deviation from a baseline response time/throughput model that are created and continuously updated through online learning [12]. Wang *et al.* proposed a dynamic causal relationship analysis approach to construct impact graphs amongst applications without given the topology [14]. Yan *et al.* described the design and development of a Generic Root Cause Analysis platform (G-RCA) for service quality management (SQM) in large IP networks [19]. Ma *et al.* presented a framework, StepWise, which can detect concept drift without tuning detection threshold or per-KPI (Key Performance Indicator) model parameters in a large scale KPI streams [21].

RPCA algorithm has been successfully applied in image recognition, face recognition, signal processing, computer vision and many other aspects [22]–[28].

Li *et al.* addressed the singing voice separation problem and proposed a novel unsupervised approach based on RPCA exploiting rank-1 constraint (CRPCA) [22]. Zhang *et al.* surveyed the applications of RPCA in computer vision, image processing, and video processing [23]. Mardani *et al.* corroborated the effectiveness of RPCA in unveiling traffic anomalies across flows and time [24]. Amoozegar *et al.* proposed a new robust online subspace and anomaly tracker for anomaly detection in time-evolving networks [25]. Ding *et al.* designed a more robust and useful real-time false data injection attack detection mechanism based on RPCA applied in the supervisory control and data acquisition (SCADA) system in smart grid [26]. Mahapatra *et al.* presented a method for detecting malicious data injections in PMU measurements and the corrupted signals were recovered using a RPCA-based algorithm [27]. Song *et al.* proposed an ensemble tensor decomposition to extract weak target signal of infrared thermography videos for cracks detection [28].

In this paper, we use invocation chain anomaly analysis based on RPCA and a single indicator anomaly detection algorithm to help solve the RCA problem. In brief, our contributions can be summarized as follows:

- Our algorithm is an RCA approach composing of RPCA algorithm and various single indicator anomaly detection algorithms.
- Our algorithm gets the anomalous nodes accurately by invocation chain anomaly analysis algorithm based on RPCA, and locates the anomalous indicators of the corresponding nodes by combining various single indicator anomaly detection algorithms.
- We have proved that our algorithm ranks the root causes in the top two with 91.3% precision for the given four batches of data. Our algorithm has an average score of 0.8304 in the four batches of data.

The rest of this paper is structured as follows: The problems and solutions are described in Section II. Applications of our algorithm in practical problems is demonstrated in Section III. Section IV summarizes the conclusions and describes our future work.

II. PROBLEM DESCRIPTION AND OUR SOLUTION

In this section, we describe the problem that our method is to solve. Then, we elaborate on our project framework in detail.

A. DESCRIPTION OF THE PROBLEM

The analysis object of this challenge is a microservice application system (a real microservice application in the private cloud environment of a large operator). The data is provided by Zhejiang Mobile, including static topology data between services, invocation chain data, gold indicator data of observed services, and time-series data of underlying services (operating system, oracle, container, and middleware). All the datasets in this paper could be downloaded from: http://iops.ai/.

In this competition, we aim to carry out anomaly detection on business indicators, root cause analysis at the time of anomaly. Then we further locate and investigate the failed container or node, and finally get the failed indicators. In addition to network type failures, other types of failures are needed to be located at the level of performance indicators (non-gold indicators), *i.e.*, not only the failed containers or nodes but also the performance indicators that cause the failure.

B. OVERALL FRAMEWORK OF OUR PROJECT

The basic framework of our algorithm is as follows:

- According to the success rate and average time of esb indicators every 60 seconds, we screen potential anomalous time period.
- For the potential anomalous time period, we find out the invocation chain in the nearby time period, and analyze the anomalous nodes in the invocation chain (generation of anomalous nodes candidate set based on the RPCA-based invocation chain anomaly analysis algorithm).
- For the final anomalous nodes candidate set, we find all related cmdb_id according to the given architecture table, and perform single indicator anomaly detection (Isolation Forest algorithm, 3σ principle, *etc.*) for all single indicators corresponding to these cmdb_ids. The final output of our algorithm is the final root cause that the algorithm scores and sorts the indicators.

Figure 1 and Figure 2 respectively show the invocation chain anomaly detection frame diagram and indicator anomaly detection flowchart.

Our root cause positioning strategy is as follows: Firstly, we implement the invocation chain anomaly detection algorithm based on the RPCA algorithm, and analyze a certain number of invocation chains extracted from the fault time period. Then we get the nodes with high anomaly scores. Secondly, we invoke the indicator anomaly detection algorithm

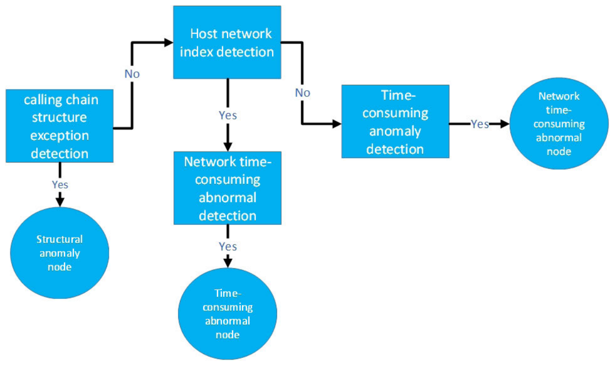

Host network index detection
No
calling chain structure exception detection
Yes
No
Time-consuming anomaly detection
Yes
Network time-consuming abnormal node
Yes
Network time-consuming abnormal detection
Structural anomaly node
Yes
Time-consuming abnormal node

FIGURE 1. Frame diagram of invocation chain anomaly detection.

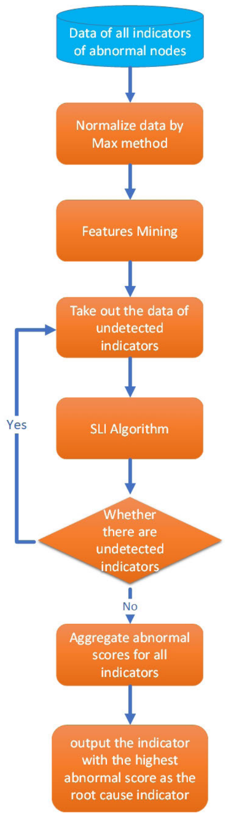

Data of all indicators of abnormal nodes
Normalize data by Max method
Features Mining
Take out the data of undetected indicators
SLI Algorithm
Yes
Whether there are undetected indicators
No
Aggregate abnormal scores for all indicators
output the indicator with the highest abnormal score as the root cause indicator

FIGURE 2. Flowchart of indicators anomaly detection.

to detect all the indicators of the anomalous nodes in the fault time period. If there is an indicator with a higher anomalous score and the indicator is also in the candidate set of root cause indicators given by the competition organizers, the indicator is the final root cause. If not, the fault type of the node is network fault.

C. ANALYSIS OF INVOCATION CHAIN ANOMALY

1) ANOMALY DETECTION STRATEGY ABOUT DIFFERENT SITUATIONS

The basic framework for anomaly detection of invocation chain nodes is as follows:

When the number of failed invocation chains in the window accounts for more than $1/3$, we carry out anomalous

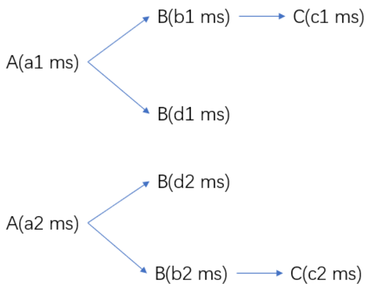


FIGURE 3. Two simple invocation chains and service time consuming.

TABLE 1. Time-consuming matrix of the two invocation chains.

0	1	2	3
a1 ms	b1 ms	c1 ms	d1 ms
a2 ms	b2 ms	c2 ms	d2 ms

ms = millisecond.

invocation chain architecture detection. Otherwise, the same window should be used for host network indicator anomaly detection. If there is a host network anomaly, we carry out network time-consuming anomaly detection. Otherwise, we carry out general time-consuming anomaly detection.

According to the three failure invocation chain modes, we propose a root cause node detection algorithm for failure invocation chain as follows:

i) We traverse every node in depth. If there is a failure node in its subtrees, we change its success status to failure.

ii) We traverse every node in depth again. If it does not contain any child node, we add it to the root cause node candidate set. If there are failure nodes in its child nodes, we do not add it to the root cause node candidate set. We continue to traverse its failure nodes' subtrees. If there are no failure nodes in its child nodes, we add all its child nodes and all nodes in its subtrees to the root cause node candidate set except the node itself.

For general time-consuming anomaly detection, there are the following strategies. We construct the standard trees by the standard tree matching algorithm. Firstly, we consider two different invocation (A, B, C are different services, ms = millisecond):

It can be seen that the structure of these two invocation chains is actually the same, so their time-consuming vectors should be located in the same time-consuming matrix. To perform anomaly detection correctly, we expect elements in each column of the time-consuming matrix to come from the nodes in the same topology location of different invocation chains, *i.e.*:

Therefore, we need an algorithm to realize nodes matching between all invocation chains. Then we briefly introduce the standard tree matching algorithm to realize this requirement. We obtain a standard tree by removing the time-consuming record in the first invocation chain, and assign a number to each node through traversal (depth-first traversal or breadth-first traversal can be used, breadth-first traversal is used in

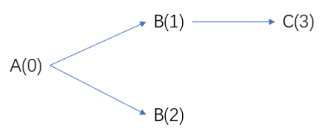


FIGURE 4. The standard tree generated by standard tree matching algorithm.

the specific implementation). The standard tree is shown in Figure 4.

Invocation chains of different topology structure correspond to different standard trees, for the given two simple invocation chains in the paper. Due to different subtree structures, the two subnodes of node *A* are distinguishable. In actual data, there is a situation where the subtrees are indistinguishable. In this case, we need to perform further subtree aggregation, superposition the same subtrees, and record the number of aggregation times for the nodes of invocation chains to ensure correctness of match.

When a new input invocation chain is processed, it is compared and matched with all the standard trees in the standard tree set. If there is a match, we obtain the corresponding standard tree node number according to the matching relationship between each node and the standard tree node. We construct the vectorized time consuming of invocation chain nodes according to the number and time consuming. After getting the time-consuming matrix, we use the RPCA algorithm to detect the anomaly of the time-consuming matrix, and obtain the same size of the anomaly submatrix. Then we output the subject corresponding to the time anomaly.

For network time-consuming anomaly detection, there are the following strategies. In the general time-consuming anomaly detection strategy, the time used by the anomaly nodes is actually the sum of the local time used by the node and the communication time used between the child nodes. If the fault is about the network and originates from the child nodes, the corresponding communication time consuming anomaly will be reflected in the time used by its parent node. Therefore, we assign the time used by the child nodes as the sum of the local time used by the parent nodes and the communication time used between the child nodes. It is expected that the fault nodes will be detected correctly in case of network failure.

For the combination of general time-consuming anomaly detection and network time-consuming anomaly detection, we can find the following rules:

i) The shallower the anomalous time is, the more obvious it is.
ii) The subject often has a parent-child relationship with itself.
iii) In case of network failure, the anomalous time consumption is reflected in the last node of the failure subject.

Therefore, if the time used by the child node is taken as the sum of the time used by the parent node and communication time for both general faults and network faults, the anomalous subject can still be correctly detected in general faults due to the rules i) and ii).

2) RPCA ALGORITHM

The basic principle of invocation chain node anomaly detection based on RPCA is as follows.

The core idea of the algorithm is that for the invocation chains in a period of time, we inspect the standard trees of the invocation chains. We calculate those nodes of the standard tree corresponding to the internal nodes of each invocation chain. Suppose that each invocation chain views the same node as the start node, then the first node of each invocation chain can be thought corresponding to the root node of the standard tree. By this means, each node of the standard trees would get a time-consuming series by calculation. The anomalous degree of time-consuming series for each standard tree can be inspected by RPCA algorithm. Finally, the algorithm would return the candidate set of anomalous nodes and corresponding anomalous coefficients in a given number of standard trees.

In the data of invocation chains, there is a condition that the same node completing the same call many times. For example, the service csf_001 calls local_method_004 four times and every local_method_004 has carried on the jdbc access to the db_009. For different invocation chains, these local_method_004 could not be distinguished from each other. Hence, it's needed to aggregate these nodes that can not be distinguished in the analysis of anomaly, then add up the time consuming of these nodes, this operation is called as subtree aggregation. In other words, all the same subtrees under the same node could be aggregated as a new subtree, whether the subtree is the same or not is defined under the aggregation of subtrees. In addition, the definition is recursive [13], [34].

The strategy of the calculation of the candidate sets of anomalous nodes in standard trees and anomalous coefficients is as follows.

Firstly, the time series of each node in the standard tree is taken as a column vector to obtain the time-consuming matrix. What's more, the columns of time-consuming matrix are standardized to ensure the adaptability of the algorithm to data of different orders of magnitude. Then RPCA algorithm would be carried out on the processed time-consuming matrix. For a given k value, the larger first k characteristic value would be selected. For each selected principal components, the absolute value of the largest element in linear transformation weight vector corresponding to the nodes as the main component of the corresponding node. The anomalous coefficients of the nodes are defined as characteristics of principle components multiplied by the node weights corresponding to the weight vectors in principle component linear transformation.

The brief introductions of RPCA algorithm are as follows.

RPCA algorithm: RPCA (robust principal component analysis) is essentially a problem of finding the best projection of data in low dimensional space like the classical PCA algorithm [33]. When the observation data is large enough, PCA algorithm can not give ideal results, while RPCA algorithm can recover the data with essentially low rank from the large and sparse noise polluted observation data. RPCA algorithm considers such a problem: The general data matrix D contains both structural information and noise. Then the matrix can be decomposed into two matrices and added: $D = A + E$, A is low rank (due to the internal structure information, there is a linear correlation between rows or columns), E is sparse (if there is noise, it is sparse), then RPCA algorithm can be written as the following optimization problem [34]–[37], *i.e.*:

$$\begin{cases} \min\limits_{A,E} \; rank(A) + \lambda ||E||_0, \\ s.t. \quad A + E = D. \end{cases} \tag{1}$$

Since rank and L_0 norm have non convex and non smooth properties, this NP problem is generally transformed into solving a relaxed convex optimization problem:

$$\begin{cases} \min\limits_{A,E} \; ||A||_* + \lambda ||E||_1, \\ s.t. \quad A + E = D. \end{cases} \tag{2}$$

D. INDICATORS ANOMALY DETECTION

In this section, we would like to introduce our single indicator anomaly detection algorithm in detail. The algorithm is composed of two parts: The first part is Features Mining Method and the second part is SLI Algorithm, which ingrates One-Class Support Vector Machine, Local Outlier Factor algorithm, Isolation Forest algorithm, and 3σ principle. SLI algorithm can combine with the aforementioned features mining method and make full use of the advantages of IF algorithm, One-Class SVM, LOF algorithm and 3σ principle.

1) FEATURES MINING METHOD

In order to eliminate the influence of dimension and ensure that the scale of data change is not lost, we first use the Max method to normalize KPI data of the anomalous node as follow, *i.e.*,

$$x_i = \frac{X_i}{X_{max}}, \tag{3}$$

where $X = (X_1, X_2, \ldots, X_n)$ represents a KPI vector, $X_{max} = max\{X_1, X_2, \ldots, X_n\}$, n is the dimension of the KPI vector X. In order to make full use of the information in the indicator data, six features [38] of the normalized KPI data $x = (x_1, x_2, \ldots, x_n)$ are mined as input data for the subsequent anomaly detection algorithm (SLI Algorithm). Different from [38], we consider both local and global features of the data in feature extraction.

$$\begin{cases} y_i^{(1)} = x_i, \\ y_i^{(2)} = x_i - x_{i-1}, \\ y_i^{(3)} = x_i - 2x_{i-1} + x_{i-2}, \\ y_i^{(4)} = x_i - 3x_{i-1} + 3x_{i-2} + x_{i-3}, \\ y_i^{(5)} = \dfrac{1}{m} \sum\limits_{j=i-m}^{i} x_j, \\ y_i^{(6)} = \sqrt{\dfrac{1}{m} \sum\limits_{j=i-m}^{i} (x_j - y_i^{(5)})^2}, \end{cases} \tag{4}$$

where x_j, $j = i - m, i - m + 1, \ldots, i$ represent the KPI data five minutes before the corresponding time point of x_i, and m represents the amount of data five minutes before the corresponding time point t_i of x_i.

2) SLI ALGORITHM

Firstly, we briefly introduce the four algorithms Isolation Forest, One-Class SVM, LOF algorithm and 3σ principle as follows.

a: ISOLATION FOREST

In an isolated forest, an anomaly is defined as more likely to be separated, which is understandable as a point thinly distributed and far from a dense population. In an isolated forest, the data set is recursively randomly divided until all the sample points are isolated. Under this random segmentation strategy, the anomalous point w have a short path usually and we define $L(w) = 1$ [39]. Otherwise, we define $L(w) = 0$.

b: ONE-CLASS SVM

It is assumed that the parameters of the generated hypersphere are center c and the corresponding hypersphere radius $r > 0$, the volume of the hypersphere $F(r)$ is minimized and center c is a linear combination of the lines that are supported. Similar to the traditional SVM method, the distance between all training data points y_i and the center can be required to be strictly less than r. But for better performance, penalty coefficient D of the relaxation variable ϑ is constructed [40], [41]. Thus, the optimization problem can be written as follows, *i.e.*,

$$\begin{cases} \min\limits_{r,o} F(r) + \sum\limits_{j=1}^{n} \vartheta_i, \\ ||y_i - c||_2 \le r + \vartheta_i, \quad i = 1, 2, 3, \ldots, n, \\ \vartheta_i \ge 0, \quad i = 1, 2, 3, \ldots, n. \end{cases} \tag{5}$$

Next, we use Lagrange dual to solve the above optimization problem (5). For the new data point p, if the distance from p to the center is less than or equal to the radius r, it's not an anomalous point and we define $L(p) = 0$, otherwise it's an anomalous point and we define $L(p) = 1$.

c: LOF ALGORITHM

The relevant expressions of the LOF algorithm are defined as follows, *i.e.*,

$$d(w, a) = ||w - a||_2, \tag{6}$$

where $d(w, a)$ represents the distance from w to a. What's more, $d_k(w)$ is defined as the k-distance of w. That is to say $d_k(w)$ represents the distance from the kth most distant point of w to w and k is a constant. $O_k(w)$ is defined as k-distance neighborhood of w, which represents the set of all the points which are less than $d_k(w)$ away from w. dr is defined as the kth reach-distance from point w to point a, which satisfies the following expression [42].

$$dr - d_k(w, a) = max(k - d(w), d(w, a)). \tag{7}$$

Furthermore, the local reachability density is defined as follows, *i.e.*,

$$dl_k(w) = \frac{|O_k(w)|}{\sum_{a\in O_k(w)} dr - d_k(w, a)}. \tag{8}$$

The local outlier factor is defined as follows, *i.e.*,

$$lof_k(w) = \frac{\sum_{a\in O_k(w)} \frac{dl_k(a)}{dl_k(w)}}{|O_k(w)|} = \frac{\sum_{a\in O_k(w)} \frac{dl_k(a)}{dl_k(w)}}{|O_k(w)|dl_k(w)}. \tag{9}$$

For LOF algorithm, if the $lof_k(w)$ is less than 1, it means that the density of w is higher than the density of its neighborhood points, and w is the normal point and we define $L(w) = 0$. If the $lof_k(w)$ is greater than 1, it means that the density of w is less than the density of its neighborhood points, and the more likely that w is the anomalous point and and we define $L(w) = 1$.

d: 3σ PRINCIPLE

We define

$$v_i = |\frac{x_i - x_{mean}}{x_{std}}|, \tag{10}$$

where x_i represents the ith data point, $x = [x_1, x_2, \ldots, x_n]$, $x_{mean} = \sum_i^n x_i/n$ and $x_{std} = \left[\sum_{i=1}^n (x_i - x_{mean})^2 /n\right]^{1/2}$. If $v_i > 3$, x_i can be viewed as an anomalous point and we define $L(w) = 1$. Otherwise, we define $L(w) = 0$.

Secondly, A detailed description of the SLI algorithm is as follows, *i.e.*,

INPUT:

When training the model, we input $T = [T_1, T_2, T_3, T_4, T_5, T_6]$ and

$$\begin{cases} T_1 = [y_1^{(1)}, y_2^{(1)}, \ldots, y_n^{(1)}]^T, \\ T_2 = [y_1^{(2)}, y_2^{(2)}, \ldots, y_n^{(2)}]^T, \\ T_3 = [y_1^{(3)}, y_2^{(3)}, \ldots, y_n^{(3)}]^T, \\ T_4 = [y_1^{(4)}, y_2^{(4)}, \ldots, y_n^{(4)}]^T, \\ T_5 = [y_1^{(5)}, y_2^{(5)}, \ldots, y_n^{(5)}]^T, \\ T_6 = [y_1^{(6)}, y_2^{(6)}, \ldots, y_n^{(6)}]^T, \end{cases} \tag{11}$$

where T represents the data obtained from the original training data X_{train} through the features mining method. X_{train} represents the data before the anomalous time period of the corresponding indicator.

When we verify the algorithm, we input $U = [U_1, U_2, U_3, U_4, U_5, U_6]$, where U represents the data obtained from the original data X_{test} through the features mining method. X_{test} is an indicator data of the anomalous node in the anomalous period.

OUTPUT:

The anomalous score $S = q_{ab}/q_{all}$ of the indicator in the anomalous period, where q_{all} represents the amount of indicator data of the anomalous node in the anomalous period, q_{ab} represents the number of anomalous data points detected by the SLI algorithm.

The whole steps of the SLI algorithm are described as follows.

Step 1: Normalize KPI data of the anomalous node by the Max method.

Step 2: Use features mining method to extract the features of normalized KPI data.

Step 3: Use training data to train the Isolation Forest, One-Class SVM, LOF algorithm and 3σ principle.

Step 4: Use four models trained in step 3 to detect the data of anomalous subjects in anomalous time period.

Step 5: Calculate $N(x_i) = L_{if}(x_i) + L_{svm}(x_i) + L_{lof}(x_i) + 2L_{sigma}(x_i)$, where $L_{if}(x_i)$, $L_{svm}(x_i)$, $L_{lof}(x_i)$ and $L_{sigma}(x_i)$ represent the label value of x_i obtained by Isolation Forest algorithm, One-Class SVM, LOF algorithm and 3σ principle respectively. If $N(x_i) > 3$, we take $L(x_i) = 1$ and x_i is judged as an anomalous point.

Step 6: Calculate the anomalous score S of the indicator in the anomalous period.

III. EXPERIMENT AND RESULT ANALYSIS

This section mainly aims to verify the effectiveness of our algorithm framework. In this section, firstly, we select the sample data and the first three batches of test data of the 2020 International AIOps Challenge to evaluate our algorithm. Secondly, we compare the root cause analysis results of IF, LOF and One-Class SVM algorithm with our algorithm. Finally, we analyze the results of our algorithm in detail.

A. PERFORMANCE OF OUR ALGORITHM

Firstly, we should read the file containing all invocation chain information given by the competition organizers. Then, we define the invoking relationship and establish complete invocation chains according to three reserved fields 'id', 'traceid', and 'pid' of the file. Because the complete invoking topology diagram is too large, part of the complete invoking topology diagram is shown in Figure 5.

Anomalies are usually observed in several monitoring indicators. For microservices, response and latency times are important key performance indicators (KPIs). We extract more static attributes or KPIs from other subjects, such as CPU memory usage, container memory usage and cache-related KPIs.

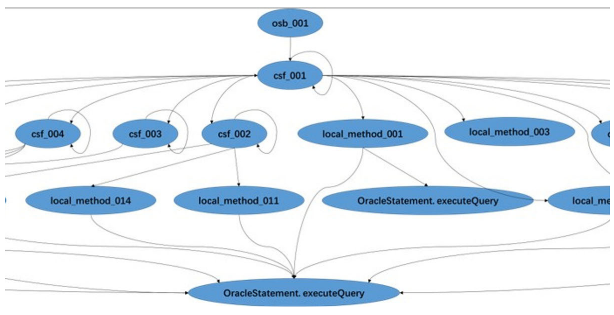


FIGURE 5. Invoking relationship topology diagram.

TABLE 2. Main monitoring KPIs.

Type of Resources	KPIs	Description
Container	container_mem_used	MEM usage(%)
	container_cpu_used	CPU usage(%)
Operating system	Sent_queue	Send queue
	Received_queue	Receive queue
Oracle	Proc_User_Uesd_Pct	Percentage of user connection usage(%)
	Proc_Uesd_Pct	Percentage of total connection usage(%)
	Sess_Connect	Number of connected sessions
	On_Off_State	Judge whether to connect or not
	tnspring_result_time	Response time of tnspring(ms)

ms=millisecond.

In the preliminary stage of this competition, the competition organizers inject the fault into the target environment by manual injection, and the duration of the injection is 6 hours (from 0:00 a.m. to 6:00 a.m. every day). Since different technology stacks have different indicators and the number of effective indicators reaches more than 100, a large number of resources and costs will be wasted if all of them are detected. By testing, some specific KPIs are collected as shown in Table 2.

Through the test and analysis of the sample data given by the competition organizers, we filter hundreds of KPI data, and finally get some important KPI data shown in Table 2. These KPI data has an obvious data fluctuation in the time period of failure, which facilitates our anomaly detection algorithm to identify the specific fault container. We simplify the set of root cause indicators to effectively reduce the computational complexity of anomaly detection algorithm and improve the overall efficiency of the algorithm, to more accurately locate the specific anomalous indicators. We eliminate some KPI data which change little with time, and use certain prior knowledge to simplify the detection complexity.

The fault time information of the sample data has been provided by the competition organizers, which is convenient for the teams to debug their own algorithm. From Figure 6, we see that during the fault period (5 minutes), the value of KPI curve of business indicators will fluctuate significantly, and accordingly, some indicators related to the underlying services will be anomalous during this period of time.

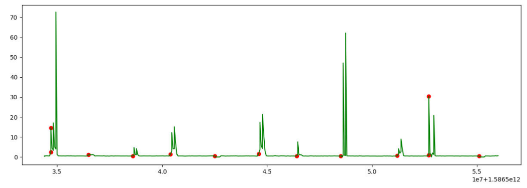


FIGURE 6. Average response time for business indicator data. The time nodes where the fault period begins is marked in red.

TABLE 3. 11 faults of the sample data.

Index	Time	Actual Fault	Detected Fault	Anomalous Indicator
1	2020/4/11 0:05	docker_003	docker_003 os_019	container_cpu_used
2	2020/4/11 0:35	docker_002	docker_002 os_018	NULL
3	2020/4/11 1:10	docker_001	docker_001 os_017	NULL
4	2020/4/11 1:40	docker_004	docker_004 os_020	container_cpu_used
5	2020/4/11 2:15	db_007	db_007	Proc_User_Used_Pct Proc_Used_Pct Sess_Connect
6	2020/4/11 2:50	docker_002	docker_002 os_018	container_cpu_used
7	2020/4/11 3:20	docker_003	docker_003 os_019	NULL
8	2020/4/11 3:55	docker_003	docker_003 os_019	NULL
9	2020/4/11 4:40	docker_008	docker_008 os_020	container_cpu_used
10	2020/4/11 5:05	docker_007	docker_002 os_018	NULL
11	2020/4/11 5:45	db_003	db_003	On_Off_State tnsping_result_time

Each fault lasts for 5 minutes. The value of anomalous indicator is 'NULL' indicates that the fault type is network fault.

TABLE 4. Partial application deployment architecture list.

Type	Number	Host Computer	Name
container_001	4	os_017	docker_001
		os_018	docker_002
		os_019	docker_003
		os_020	docker_004
container_002	4	os_017	docker_005
		os_018	docker_006
		os_019	docker_007
		os_020	docker_008

We debug our algorithm with the sample data. Firstly, anomalous nodes of 10 out of 11 faults are located by our invocation chain anomaly analysis algorithm based on RPCA algorithm. Then, specific fault nodes of corresponding anomalous containers could also be verified accurately by our single indicator anomaly detection algorithm. By comparing the output candidate set of root cause and the file of fault content given by the competition organizers, the top two root causes located by our algorithm could hit the real root cause. According to table 3, it can be seen that for the detected faults of two anomalous nodes, take the first fault as an example. Because docker_003 is deployed on os_019, the invocation chain detection algorithm would get two anomalous nodes, *i.e.*, the anomalous node itself and its deployed principle. Through the experiment of the sample data, it can be seen that our algorithm has a higher accuracy to some extent. Table 4 shows a partial application deployment architecture list.

TABLE 5. 11 faults of the first batch of test data.

Index	Time	Actual Fault	Detected Fault	Anomalous Indicator
1	2020/4/22 0:05	docker_004	docker_004 os_020	container_cpu_used
2	2020/4/22 0:40	db_003	db_003	Proc_User_Used_Pct Proc_Used_Pct Sess_Connect
3	2020/4/22 1:16	os_021	os_021	Sent_queue Received_queue
4	2020/4/22 1:52	os_009 (flyremote)	flyremote_001	Sent_queue Received_queue
5	2020/4/22 2:25	docker_004	docker_004 os_020	NULL
6	2020/4/22 3:00	docker_006	docker_006 os_018	NULL
7	2020/4/22 3:36	os_021	os_021	Sent_queue Received_queue
8	2020/4/22 4:12	docker_001	docker_001 os_017	NULL
9	2020/4/22 4:47	docker_006	docker_006 os_018	container_cpu_used
10	2020/4/22 5:20	docker_005	docker_005 os_017	NULL
11	2020/4/22 5:55	docker_007	docker_007 os_019	NULL

Each fault lasts for 5 minutes. The value of anomalous indicator is 'NULL' indicates that the fault type is network fault.

TABLE 6. 13 faults of the second batch of test data.

Index	Time	Actual Fault	Detected Fault	Anomalous Indicator
1	2020/4/23 1:47	os_020	docker_004 os_020	Sent_queue Received_queue
2	2020/4/23 3:47	os_017	docker_005 os_017	Sent_queue Received_queue
3	2020/4/23 4:47	db_003	db_003	On_Off_State tnsping_result_time
4	2020/4/26 0:13	os_021	os_021	Sent_queue Received_queue
5	2020/4/26 0:43	docker_005	docker_004 os_020	NULL
6	2020/4/26 1:43	docker_006	docker_006 os_018	NULL
7	2020/4/26 2:13	docker_001	docker_001 os_017	NULL
8	2020/4/26 2:43	os_009 (flyremote)	flyremote_001	Sent_queue Received_queue
9	2020/4/26 3:43	os_020	os_020	Sent_queue Received_queue
10	2020/4/26 4:13	docker_002	docker_002 os_018	contain_cpu_used
11	2020/4/26 4:43	docker_004	docker_004 os_020	NULL
12	2020/4/26 5:13	os_018	os_018	Sent_queue Received_queue
13	2020/4/26 5:43	docker_002	docker_002 os_018	NULL

Each fault lasts for 5 minutes. The value of anomalous indicator is 'NULL' indicates that the fault type is network fault.

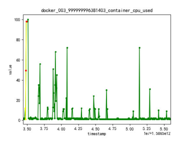
The sample data of this competition has given the specific anomalous nodes and corresponding indicators when the fault occurs. Then, we select the first three batches of test data to test our algorithm. The test data only gives the time point where the fault occurs, and contestants do not need to carry out anomaly detection on the esb indicator. The given time points to the time stamp are taken as the input of our algorithm to get the final anomalous nodes and corresponding anomalous indicators, as shown in Tables 5–7.

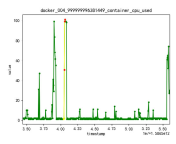
According to Table 5, we consider the 4*th* fault in the first batch of test data. The actual fault is located on os_009,

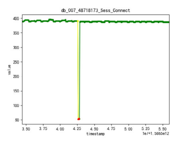
TABLE 7. 11 faults of the third batch of test data.

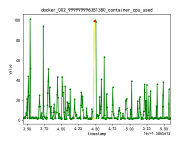
Index	Time	Actual Fault	Detected Fault	Anomalous Indicator
1	2020/4/21 0:17	docker_007	docker_007 os_19	NULL
2	2020/4/21 0:47	docker_001	docker_001 os_017	NULL
3	2020/4/21 1:47	docker_008	docker_001 os_017	NULL
4	2020/4/21 2:47	docker_008	docker_008 os_020	NULL
5	2020/4/21 2:47	os_021	os_021	Sent_queue
6	2020/4/21 3:17	os_017	os_017	Sent_queue
7	2020/4/21 3:47	db_003	db_003	Proc_User_Used_Pct Proc_Used_Pct Sess_Connect
8	2020/4/21 4:17	docker_004	docker_004 os_017	container_cpu_used
9	2020/4/21 4:47	docker_006	docker_006 os_018	NULL
10	2020/4/21 5:17	docker_006	docker_004 os_020	NULL
11	2020/4/21 5:48	docker_003	docker_003 os_019	NULL

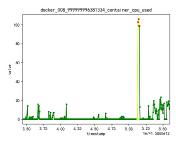
Each fault lasts for 5 minutes. The value of anomalous indicator is 'NULL' indicates that the fault type is network fault.


(a) Fault 1: Container_cpu_used of docker_003.


(b) Fault 4: Container_cpu_used of docker_004.


(c) Fault 5: Sess_Connect of db_007.


(d) Fault 6: Container_cpu_used of docker_002.


(e) Fault 9: Container_cpu_used of docker_008.

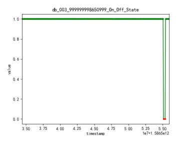
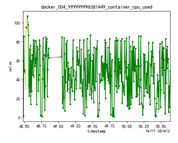
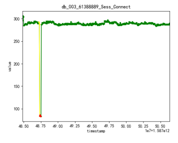

(f) Fault 11: On_Off_State of db_003.

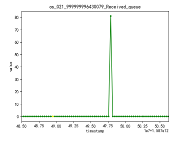
FIGURE 7. KPI curves corresponding to anomalous nodes detected in the fault time period except network fault of sample data. Green lines show normal time periods. Yellow lines show anomalous time periods. The anomalous points detected by our indicator anomaly detection algorithm are marked in red.

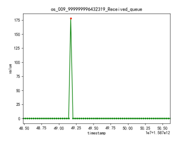
actually is fly remote according to application deployment architecture list given by competition organizers. The specific reason is that the complete set of faults given by the competition organizers does not include faults of fly remote. By manually viewing the failed invocation chain information and the KPI curves of the anomalous indicators, we could lock the location of anomalous node as flyremote_001.

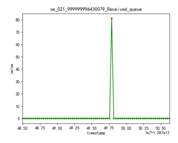
Figures 7-10 show KPI curves corresponding to anomalous nodes detected in the fault time period except network fault of the first three batches of test data. From our root cause location strategy mentioned in Section II, we can see that there is no corresponding anomalous indicator of network fault. If the anomalous node is docker, it is generally the CPU fault and the value of the indicator 'container_cpu_used' is usually around 90, for example, Figure 7(a) and 8(a). If the anomaly is db, there are generally two types of faults:


(a) Fault 1: Container_cpu_used of docker_004.


(b) Fault 2: Sess_Connect of db_003.


(c) Fault 3: Received_queue of os_021.


(d) Fault 4: Received_queue of os_009.


(e) Fault 7: Received_queue of os_021.

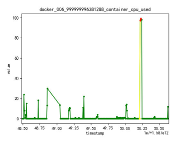
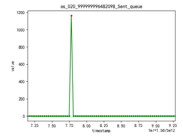
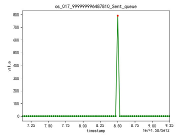
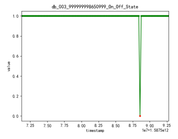
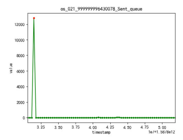

(f) Fault 9: Container_cpu_used of docker_006.

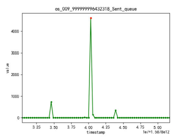
FIGURE 8. **KPI curves corresponding to anomalous nodes detected in the fault time period except network fault of the first batch of test data. Green lines show normal time periods. Yellow lines show anomalous time periods. The anomalous points detected by our indicator anomaly detection algorithm are marked in red.**

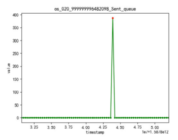

(a) Fault 1: Sent_queue of os_020.

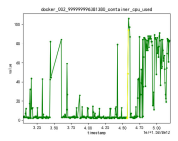

(b) Fault 2: Sent_queue of os_017.


(c) Fault 3: ON_Off_State of db_003.


(d) Fault 4: Sent_queue of os_021.


(e) Fault 8: Sent_queue of os_009.


(f) Fault 9: Sent_queue of os_020.


(g) Fault 10: Container_cpu_used of docker_002.

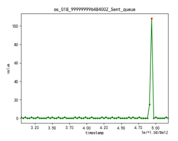
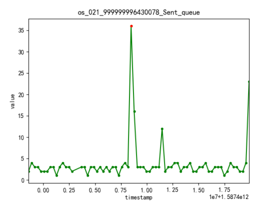
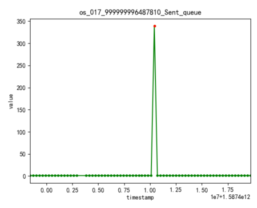
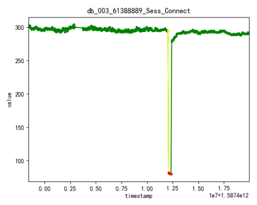

(h) Fault 12: Sent_queue of os_018.

FIGURE 9. **KPI curves corresponding to anomalous nodes detected in the fault time period except network fault of the second batch of test data. Green lines show normal time periods. Yellow lines show anomalous time periods. The anomalous points detected by our indicator anomaly detection algorithm are marked in red.**

database connection limit and database off monitoring. For the former, the indicator 'Sess_Connection' would drop to a very low value, for example, Figure 7(c) and 8(b). For the latter, the indicator 'On_Off_State' would change from 1 to 0, for example, Figure 7(f) and 9(c). If the anomaly is os, the values of indicators 'Sent_queue' and 'Received_queue' would increase sharply, and the indicator 'Received_queue' would show a certain lag, for example, Figure 8(c) and 9(b).


(a) Fault 5: Sent_queue of os_021.


(b) Fault 6: Sent_queue of os_017.


(c) Fault 7: Sess_Connect of db_003.

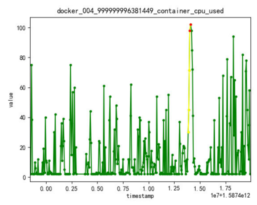

(d) Fault 8: Container_cpu_used of docker_004.

FIGURE 10. **KPI curves corresponding to anomalous nodes detected in the fault time period except network fault of the third batch of test data. Green lines show normal time periods. Yellow lines show anomalous time periods. The anomalous points detected by our indicator anomaly detection algorithm are marked in red.**

TABLE 8. **Comparative results of different algorithms.**

Data	IF Algorithm	LOF Algorithm	One Class SVM Algorithm	Our Algorithm
Sample data	0.8181	0.5090	0.6545	0.8363
1^{st} test data	0.6184	0.6000	0.6184	0.8545
2^{nd} test data	0.6462	0.6307	0.5846	0.7846
3^{rd} test data	0.6307	0.5230	0.5076	0.8461
Average	0.6784	0.5657	0.5913	0.8304

B. PERFORMANCE OF DIFFERENT ALGORITHMS

Table 8 shows the comparison results on the sample data and the first three batches of data of the 2020 International AIOps Challenge using different algorithms. We transform the fault time given by each batch of data into the corresponding timestamp as the input. We obtain the final root cause indicator score by indicator anomaly detection with four algorithms, IF algorithm, LOF algorithm, One-Class SVM algorithm and our algorithm. And our indicator anomaly detection algorithm is the combination of voting scores of three algorithms IF, LOF, and One-Class SVM. Besides, the final scoring rules of the root cause indicator are consistent with the rules of this International AIOps Challenge, *i.e.*, if the first root cause indicator is correct, we get 1 point, if the second root cause indicator is correct, we get 0.2 point, if there is no root cause indicator is correct, we get 0 point. The total score of root cause indicators divided by the amount of faults is the final root cause indicator score.

Compared with IF algorithm (The average score of the four batches of data is 0.6784), LOF algorithm (The average score of the four batches of data is 0.5657), and One-Class SVM algorithm (The average score of the four batches of data is 0.5913), our algorithm (The average score of the four batches of data is 0.8304) has a higher score, which indicates our single indicator algorithm including three algorithms has a better root cause location effect.

IV. CONCLUSION

In this paper, we propose a root cause location method based on execution trace for MSA. Firstly, we describe the execution trace of request processing by invocation trees, and construct a standard tree library by using the standard tree matching algorithm. Then, we extract the invocation chains in the period of partial execution trace failure to match with the standard tree library, and vectorize the time-consuming matrix of the standard tree nodes to build an anomalous time-consuming matrix. We use RPCA algorithm to analyze the anomalous time-consuming matrix, and the single indicator anomaly detection algorithm to locate the root causes. Finally we output nodes with high anomalous scores. Experimental results show that the proposed algorithm ranks the root causes in the top two with over 90% precision for given sample data and test data, which verify the effectiveness of the proposed algorithm.

At present, this method still has the following problems to be improved: Firstly, the root cause location of offline data is based on the condition that the competition organizers have given the specific time nodes of fault occurrence. Our next work is to explore some effective anomaly detection algorithms for esb indicators; Secondly, our algorithm only performs well for offline data. To successfully apply our algorithm to online data, we try to use Kafka to accept streaming data from the cloud services. After we convert the received streaming data into the data in a specific format, the online data can be perfectly connected with our algorithm.

REFERENCES

[1] Á. Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, and V. Muntés-Mulero, "Graph-based root cause analysis for service-oriented and microservice architectures," *J. Syst. Softw.*, vol. 159, pp. 110432.1–110432.17, Jan. 2020.

[2] J. Zhao and L. Huang, "Exploration of research on microservice fault diagnosis technology," *J. Netw. New Media*, vol. 9, no. 1, pp. 57–64, Jan. 2020.

[3] W. Lin, M. Ma, D. Pan, and P. Wang, "FacGraph: Frequent anomaly correlation graph mining for root cause diagnose in micro-service architecture," in *Proc. IEEE 37th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Nov. 2018, pp. 1–8.

[4] J. M. Q. Alvarez, J. A. O. Sanabria, and J. I. M. Garcia, "Microservices-based architecture for fault diagnosis in tele-rehabilitation equipment operated via Internet," in *Proc. IEEE Latin Amer. Test Symp. (LATS)*, Mar. 2019, pp. 1–6.

[5] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Trans. Softw. Eng.*, early access, Dec. 18, 2018, doi: 10.1109/TSE.2018.2887384.

[6] L. Huang, W. Zhuang, M. Sun, and H. Zhang, "Research and application of microservice in power grid dispatching control system," in *Proc. IEEE 4th Inf. Technol., Netw., Electron. Automat. Control Conf. (ITNEC)*, Jun. 2020, pp. 1895–1899.

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

[8] Z. Wang, T. Wang, W. Zhang, N. Chen, and C. Zuo, "Fault diagnosis for microservices with execution trace monitoring," *J. Softw.*, vol. 28, no. 6, pp. 1435–1454, Jun. 2017.

[9] M. Ma, Z. Yin, S. Zhang, S. Wang, C. Zheng, X. Jiang, H. Hu, C. Luo, Y. Li, N. Qiu, F. Li, C. Chen, and D. Pei, "Diagnosing root causes of intermittent slow queries in cloud databases," *Proc. VLDB Endowment*, vol. 13, no. 10, pp. 1176–1189, Jun. 2020.

[10] X. Nie, Y. Zhao, K. Sui, D. Pei, Y. Chen, and X. Qu, "Mining causality graph for automatic Web-based service diagnosis," in *Proc. IEEE 35th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Dec. 2016, pp. 1–8.

[11] L. Meng, Y. Sun, and S. Zhang, "Midiag: A sequential trace-based fault diagnosis framework for microservices," in *Proc. 17th Int. Conf. Services Comput. (SCC)*, Honolulu, HI, USA, Sep. 2020, pp. 137–144.

[12] J. Xu, Y. Wang, P. Chen, and P. Wang, "Lightweight and adaptive service API performance monitoring in highly dynamic cloud environment," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Honolulu, HI, USA, Jun. 2017, pp. 35–43.

[13] P.-S. Huang, S. D. Chen, P. Smaragdis, and M. Hasegawa-Johnson, "Singing-voice separation from monaural recordings using robust principal component analysis," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2012, pp. 57–60.

[14] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "CloudRanger: Root cause identification for cloud native systems," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2018, pp. 492–502.

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

[16] P. Chen, Y. Qi, and D. Hou, "CauseInfer: Automated end-to-end performance diagnosis with hierarchical causality graph in cloud environment," *IEEE Trans. Services Comput.*, vol. 12, no. 2, pp. 214–230, Mar. 2019.

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

[18] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "AutoMAP: Diagnose your microservice-based Web applications automatically," in *Proc. Web Conf.*, Apr. 2020, pp. 246–258.

[19] H. Yan, L. Breslau, Z. Ge, D. Massey, D. Pei, and J. Yates, "G-RCA: A generic root cause analysis platform for service quality management in large IP networks," *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1734–1747, Dec. 2012.

[20] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng, "Opprentice: Towards practical and automatic anomaly detection through machine learning," in *Proc. Internet Meas. Conf.*, Oct. 2015, pp. 211–224.

[21] M. Ma, S. Zhang, D. Pei, X. Huang, and H. Dai, "Robust and rapid adaption for concept drift in software system anomaly detection," in *Proc. IEEE 29th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Memphis, TN, USA, Oct. 2018, pp. 13–24.

[22] F. Li and M. Akagi, "Unsupervised singing voice separation based on robust principal component analysis exploiting rank-1 constraint," in *Proc. 26th Eur. Signal Process. Conf. (EUSIPCO)*, Sep. 2018, pp. 1920–1924.

[23] T. Bouwmans, S. Javed, H. Zhang, Z. Lin, and R. Otazo, "On the applications of robust PCA in image and video processing," *Proc. IEEE*, vol. 106, no. 8, pp. 1427–1457, Aug. 2018.

[24] M. Mardani, G. Mateos, and G. B. Giannakis, "Recovery of low-rank plus compressed sparse matrices with application to unveiling traffic anomalies," *IEEE Trans. Inf. Theory*, vol. 59, no. 8, pp. 5186–5205, Aug. 2013.

[25] M. Amoozegar, B. Minaei-Bidgoli, M. Rezghi, and H. Fanaee-T, "Extra-adaptive robust online subspace tracker for anomaly detection from streaming networks," *Eng. Appl. Artif. Intell.*, vol. 94, Sep. 2020, Art. no. 103741.

[26] Y. Ding and J. Liu, "Real-time false data injection attack detection in energy Internet using online robust principal component analysis," in *Proc. IEEE Conf. Energy Internet Energy Syst. Integr. (EI)*, Beijing, China, Nov. 2017, pp. 1–6.

[27] K. Mahapatra and N. R. Chaudhuri, "Malicious corruption-resilient wide-area oscillation monitoring using online robust PCA," in *Proc. IEEE Power Energy Soc. Gen. Meeting (PESGM)*, Portland, OR, USA, Aug. 2018, pp. 1–5.

[28] J. Song, B. Gao, W. L. Woo, and G. Y. Tian, "Ensemble tensor decomposition for infrared thermography cracks detection system," *Infr. Phys. Technol.*, vol. 105, Mar. 2020, Art. no. 103203.

[29] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 1, pp. 93–104, Jun. 2013.

[30] N. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, Aug. 2015., pp. 1939–1947.

[31] W. Xu, L. Huang, A. Fox, D. A. Patterson, and I. M. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, Haifa, Israel, Jun. 2010, pp. 112–132.
[32] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Proc. 9th IEEE Int. Conf. Data Mining*, Miami, FL, USA, Dec. 2009, pp. 149–158.
[33] M. Partridge and M. Jabri, "Robust principal component analysis," in *Proc. IEEE Signal Process. Soc. Workshop Neural Netw. Signal Process.*, Sydney, NSW, Australia, vol. 1, Dec. 2000, pp. 1–10.
[34] E. J. Candes, X. Li, Y. Ma, and J. Weight, "Robust principal component analysis?" *J. ACM*, vol. 58, no. 3, pp. 1–37, Dec. 2009.
[35] J. Wright, A. Ganesh, S. Rao, and Y. Ma, "Robust principal component analysis: Exact recovery of corrupted low-rank matrices," Apr. 2009, *arXiv:0905.0233*. [Online]. Available: https://arxiv.org/abs/0905.0233
[36] N. Vaswani, T. Bouwmans, S. Javed, and P. Narayanamurthy, "Robust subspace learning: Robust PCA, robust subspace tracking, and robust subspace recovery," *IEEE Signal Process. Mag.*, vol. 35, no. 4, pp. 32–55, Jul. 2018.
[37] N. Vaswani, Y. Chi, and T. Bouwmans, "Rethinking PCA for modern data sets: Theory, algorithms, and applications [scanning the issue]," *Proc. IEEE*, vol. 106, no. 8, pp. 1274–1276, Aug. 2018.
[38] Zhang, Zhu, Li, Wang, and Guo, "Anomaly detection based on mining six local data features and BP neural network," *Symmetry*, vol. 11, no. 4, p. 571, Apr. 2019.
[39] F. Liu, K. Ting, and Z. Zhou, "Isolation forest," in *Proc. IEEE 8th Int. Conf. Data Mining*, Pisa, Italy, Dec. 2008, pp. 413–422.
[40] R. Perdisci, G. Gu, and W. Lee, "Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems," in *Proc. 6th Int. Conf. Data Mining (ICDM)*, Hong Kong, Dec. 2006, pp. 488–498.
[41] Y.-S. Choi, "Least squares one-class support vector machine," *Pattern Recognit. Lett.*, vol. 30, no. 13, pp. 1236–1240, Oct. 2009.
[42] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, May 2000, pp. 93–104.

MINGXU JIN was born in China, in 1998. He received the B.S. degree in mathematics and applied mathematics from the South China University of Technology, Guangzhou, China, in 2020, where he is currently pursuing the master's degree in computational mathematics. His research interests include computer-aided geometric design and machine learning.

AORAN LV was born in China, in 1999. He is currently pursuing the B.S. degree in information and computing science with the South China University of Technology, Guangzhou, China. His research interests include computer graphics and machine learning.

YUANPENG ZHU was born in China, in 1984. He received the master's degree in computational mathematics and the Ph.D. degree in applied mathematics from Central South University, Changsha, China, in 2011 and 2014, respectively. From 2014 to 2016, he was a Postdoctoral Fellow with the School of Mathematical Sciences, University of Science and Technology of China, Hefei, China. Since 2016, he has been with the South China University of Technology, Guangzhou, China, where he is currently a Lecturer with the the School of Mathematics. His research interests include computer-aided geometric design, geometric modeling, and spline.

ZIJIANG WEN was born in China, in 1991. He received the B.S. degree in statistics from Guangdong Medical University, Guangzhou, China, in 2015. He is currently engaged in machine learning with JOYY Inc. His research interests include artificial intelligence and deep learning.

YUBIN ZHONG was born in China, in 1993. He received the master's degree in control science and engineering from the Guangdong University of Technology, Guangzhou, China, in 2019. He is currently engaged in data mining with JOYY Inc. His research interests include machine learning, anomaly detection, and recommendation systems.

ZEXIN ZHAO was born in China, in 1999. He is currently pursuing the B.Eng. degree in information management and information system with the South China University of Technology, Guangzhou, China. His research interests include software engineering and machine learning.

JIANG WU was born in China, in 1999. He is currently pursuing the B.Eng. degree in information management and information system with the South China University of Technology, Guangzhou, China. His research interests include neural networks, optimization algorithm, data mining, big data processing, and robotics.

HEJIE LI was born in China, in 2000. He is currently pursuing the B.S. degree in information and computing science with the South China University of Technology, Guangzhou, China. His research interests include machine learning and data algorithm.

HANHENG HE was born in China, in 1998. He is currently pursuing the B.S. degree in information and computing science with the South China University of Technology, Guangzhou, China. His research interests include machine learning and computer vision.

FENGYI CHEN was born in China, in 1985. She received the master's degree in accounting science from the University of Macau, in 2012. She is currently pursuing the Ph.D. degree with The Hong Kong Polytechnic University. Her research interests include grey forecasting and market forecasting.