AAMR: Automated Anomalous Microservice Ranking in Cloud-Native Environment

Zekun Zhang, Bing Li, Jian Wang, Yongqiang Liu
School of Computer Science,
Wuhan University,
Wuhan, China
e-mail: jianwang@whu.edu.cn

***Abstract*— Recently, it has become a trend for developers to build applications using the microservice architecture. The functionality of each application is divided into multiple independent microservices, which are interconnected to others. With the emergence of cloud-native technologies, such as Docker and Kubernetes, developers can achieve a consistent and scalable delivery for complex software applications. However, it is challenging to diagnose performance issues in microservices due to the complex runtime environments and the numerous metrics. In this paper, we propose a novel root cause analysis approach named AAMR. AAMR firstly constructs a service dependency graph based on real-time metrics. Next, it updates the anomaly weight of each microservice automatically. Finally, a PageRank-based random walk is applied for further ranking root causes, i.e., ranking potential problematic services. Experiments conducted on Kubernetes clusters show that the proposed approach achieves a good analysis result, which outperforms several state-of-the-art methods.**

Keywords—Microservice, Anomaly detection, Root cause analysis, Cloud-native system

I. INTRODUCTION

Nowadays, microservice architectures (MSA) have become increasingly popular in large-scale software development following different computing paradigms like cloud computing, mobile computing, and edge computing. MSA-based software applications are decomposed into light-weighted, interconnected, independently deployed, and scalability-enabled microservices [1]. With the decomposition, the process of testing, deploying, and releasing becomes faster. However, as user requirements change, software code commits, and version updates become increasingly frequent. Many unexpected issues may arise, which have a significant impact on service quality and user experience. It is important for developers to figure out the root causes of system failures and mitigate them.

Traditionally, system failures are usually pinpointed by checking the log and event tracking, and then the performance issues are analyzed based on monitoring tools [2]. With the increasing scale and complexity of software, service dependencies also become increasingly complex, making these tools hard to achieve the needs of troubleshooting and diagnosis. In general, when an anomaly occurs in microservice systems, the anomaly detected is merely a symptom, and the root cause often hides from a larger underlying issue. Particularly, if a microservice becomes abnormal, e.g., response time delay or interruption of work, most of the microservices collaborated with it will be implicated. Therefore, it is necessary to detect undesirable performance problems and pinpoint the underlying anomalous microservice (root cause).

At present, the challenges of locating potential root causes are (*i*) *Large volume of metrics*: Communications between services are plenty and frequent, which cause a large volume of monitoring metrics (e.g., OpenStack exposes 17,608 metrics [3]). It is challenging to pinpoint the bottleneck from numerous and diverse metrics. (*ii*) *Different failure sources*: The failures might be caused by upstream or downstream tasks in the propagation direction. Besides, the wrong deployments and insufficient resource utilization can also cause failures. (*iii*) *Highly dynamic in runtime*: Due to the flexibility of microservices, the IP address of a microservice may dynamically change in creating a replica. The scalability of replicas further enlarges the service correlation and the complexity of locating anomalies.

Many existing works on root cause analysis have been reported. Most of these works [4-8] localize the root cause by constructing a service dependency graph (SDG) [10] based on monitored metrics. With the SDG, the anomalous microservices are commonly ranked by the similarity between back-end services and front-end services. However, services that have little impact on front-end services are missing in the diagnosis. As for metrics, parts of these works [5, 6] only use application-level metrics, which is insufficient for analysis. Some works [7, 8] consider multiple metrics while missing the key metrics ranking. To address these limitations, we propose a novel approach to detect anomalies and locate the root cause in microservice systems.

If there is an anomalous node in the service network, the nodes associated with it are likely affected. Inspired by the mRank [9] algorithm, we use adjacent nodes to represent the anomaly score of the target node. As for input, we collect multiple metrics, including system utilization and application-level metrics. Our goal is to localize the root cause and highlight the key anomalous metric, which helps developers diagnose system failures. We evaluate our approach on Kubernetes clusters and inject several common failures that occur in cloud-native systems. The results show that our approach outperforms several state-of-the-art methods in localizing accuracy. In summary, our contributions include:

- We extend the mRank algorithm for root cause analysis in microservices. Our method can automatically update the anomaly weights in SDG.

DOI reference number: 10.18293/SEKE2021-091

- We evaluate our method in a cloud-native environment. The experimental results show that our approach has higher accuracy and faster than other baseline methods on the benchmark.

The remainder of this paper is organized as follows. Related works are summarized in Section II. Section III formulates the problem. We elaborate on our proposed approach in Section IV. Experiments and evaluations are included in Section V. The conclusion and future work are given in Section VI.

II. Related Work

Root cause analysis for distributed systems has been devoted in the industry and academia for years. Existing approaches in this area can be approximately classified into four types.

Trace-based methods. Many tools and systems on end-to-end tracing like Dapper [11], Pinpoint [12], and EagleEye [13] collect the trace information. These tools can accurately record the execution path of programs and then locate the failure by detecting the source code or binary code. However, a large-scale system is usually developed by many teams with different languages over the years, and the overhead of modifying its source code is often too high [14].

Log-based methods. The system log is an important clue for analysis [2]. By parsing patterns and extracting features from event logs, Xu et al. [15, 16] built anomaly detection and identification models from historical data and used these models to analyze root causes. However, as the application flexibility increases, these methods are less effective in analyzing the anomalies in real-time.

Machine learning-based methods. Some researchers use the metrics collected as training data, instead of logs, to train models. Brandón et al. [17] constructed fault patterns from several fault injection methods. The anomalies are classified by comparing the similarity between the anomaly graph and fault patterns. Moreover, Du et al. [18] collected real-time performance data such as CPU, memory, response time, and package loss to build a model for anomaly detection. GRANO [19] created an anomaly analysis model and visualized the analysis result. But these approaches require collecting a large amount of data for model training, and these models cannot cover all anomalous patterns.

Graph-based methods. Many graph-based approaches are also proposed based on real-time performance metrics. For example, CloudRanger [6] constructed an impact graph based on the dynamic causal relationship. Microscope [5] added anomalous nodes into a candidate group and then ranked the anomalous nodes in the candidate group based on the correlation coefficients between nodes. But only application-level metrics are included in their works, which is insufficient for analysis. To solve such problems, MicroCause [20] used multi-metric and captured the sequential relationship of time series data, and MS-Rank [7] updated the weights of different metrics dynamically. These methods used forward, self, and backward random walk to heuristically locate root causes. Besides, Weng et al. [21] found that anomalies occur on both the service and physical level. MicroRCA [8] correlated anomalous performance symptoms with relevant resource utilization to represent service anomalies. However, MicroRCA cannot update the anomaly detection confidence (i.e., weights in SDG) automatically.

Similar to graph-based approaches, we also use a graph model and rank the anomalies using a random walk algorithm. In our approach, we automatically update the anomaly weights in SDG and output a two-phase ranking list that contains the anomalous nodes and metrics.

III. Problem Definition

To generalize the problem, we treat the microservice system as a "black box" that requires no domain knowledge, and the root cause analysis process is running independently. Many reasons can cause abnormal events in microservices, such as sudden increases in throughput, errors in code logic, and insufficient allocation of host resources. We refer to the process of diagnosing those anomalous nodes and the metrics responsible for the abnormal events as root cause analysis. The identification of anomalous nodes is regarded as root cause localization. We monitor the metrics change of all microservices in the system by default. These metrics are collected as a matrix in time window T. We denote the matrix as M, and M_k stands for the metrics in column k. Our objective is to identify a set of root causes V_{rc} and rank the associated metrics for each root cause. The notations used in the paper are listed in Table I.

TABLE I. Notations

Notation	Definitions
$G(V, E, W)$	Service dependency graph with weight matrix W
M, M_k	Metrics collected in T and metrics in column k
V_i, h_i	Microservice node i and the host node of V_i
P, p_{ij}	$[P]_{ij} = p_{ij}$, transition probability from V_i to V_j
RT_i	Response time series of V_i in T
Δt, T	Time unit for metric collection and the time window
V_{fe}, V_{rc}	Front-end service and root cause services
ADs, AS	The clustering result of RT_i and the anomaly score

IV. Approach Design

This section introduces the detail of the proposed root cause analysis approach.

A. Overall Framework

To address the above issues, we propose a novel root cause analysis approach named AAMR (short for Automated Anomalous Microservice Ranking). Fig. 1 shows the overall framework of AAMR, which consists of five stages:

$S1$: *Collect system and application-level metrics as the input;*

$S2$: *Detect anomalies;*

$S3$: *Construct the service dependency graph;*

$S4$: *Update the anomaly weights in SDG;*

$S5$: *Rank the anomalous nodes and metrics.*

$S1$ and $S2$ run continuously by default. Once anomalies are detected, the following stages are triggered. We discuss the components of AAMR in detail in the following parts.

B. Data Collection

Root cause analysis is based on performance metrics obtained by monitoring applications. Since a single metric is insufficient to reflect the anomalous degree [7], similar to [4, 5,

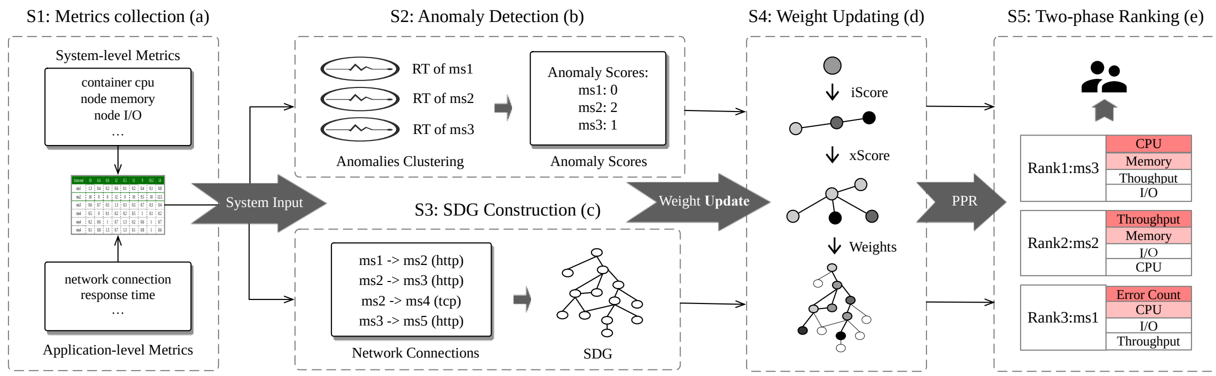

Figure 1. The overall framework of AAMR

8], we collect metrics at different levels: (*i*) *System-level Metrics*. These metrics are resource utilization metrics monitored at the physical server or virtual machine layer (e.g., CPU, memory, and network utilization of the host node). (*ii*) *Application-level Metrics*. Application-level metrics include performance metrics observed at the application layer, such as response time, workload, and network connection.

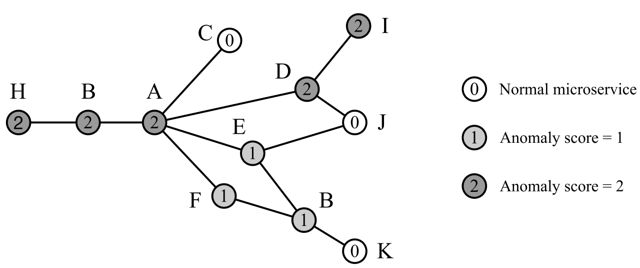

Figure 2. An example of AANs and NHANs

C. Anomaly Detection

Anomaly detection is the beginning of root cause analysis. We use the BIRCH [22] clustering algorithm for anomaly detection, which is simple but effective. We continually monitor the response time of each microservice by default. BIRCH takes the RT_i collected of each microservice in T as input. As a result, the RT_i is divided into n clusters without predefined. It is noticed that the response time of different microservices varies with different business processes. For example, if V_a handles a single business process and V_b handles compound business processes. The response time of V_a is shorter than V_b in most cases. So we cluster RT_i for each microservice instead of overall microservices. If the cluster result *ADs* of a microservice exceeds 1, it indicates this node is anomalous. Instead of simply detecting anomalies [8], we further define the anomaly score (*AS*) of this node as *ADs*-1 to represent the basic anomalous degree of each microservice.

D. Service Dependency Graph Construction

We construct a service dependency graph based on the network connection between services to represent the anomaly propagation. If service V_a sends a connection request to service V_b, we add a directed edge from V_a to V_b. As for duplicate edges, only one connection is counted to avoid redundancy. By integrating all network connections, we end up with a service dependency graph $G(V, E, W)$. It is a weighted DAG (Directed Acyclic Graph) that describes the dependency between services. Here V, E, W indicate microservice nodes, SDG edges, and the anomaly weights, respectively. Considering that some microservice connections may fail due to anomalies at the current moment, we choose the network connection details from the moment before time window T for the SDG construction.

E. Automated Anomaly Weight Updating

Once the SDG is constructed, the following processes start to locate the root cause. According to the mRank algorithm [9], if there is an anomalous node in the service network, then the nodes associated with the anomalous node are likely affected. However, it is also possible that other nodes cause the anomalies of these nodes. Therefore, to infer the possibility of a node being abnormal, we need to consider the nodes related to its neighbors. We define $AAN(V_i)$ as the anomalous-adjacent nodes of node V_i. Further, we define $NHAN(V_i)$ as the next-hop-anomalous nodes of node V_i, that is, the anomalous nodes that directly connect to $AAN(V_i)$. For example, for node A in Fig. 2, $AAN(A)$ consists of B, D, E, and F. And $NHAN(A)$ includes all the anomalous nodes that are connected to B, D, E, and F. Then we define two measurements to quantify the anomaly of a node in the following.

Definition 4.1 (*iScore*). *iScore* of a microservice V_i in SDG is defined as:

$$iScore(V_i) = \frac{\sum_{j=1}^{N} AS(V_j)}{Degree(V_i)}, V_j \in AAN\ (V_i), \tag{1}$$

where $AS(V_i)$, $Degree(V_i)$, and N denote the anomaly score of V_i, the degree of V_i, and the number of $AAN(V_i)$, respectively. As for $NHAN(V_i)$ we define:

Definition 4.2 (*xScore*). *xScore* of a microservice V_i in SDG is defined as:

$$xScore(V_i) = x(V_i) - \frac{\sum_{j=1}^{N} AS(V_j)}{\sum_{j=1}^{N} Degree(V_j)}, V_j \in NHAN(\ V_i), \tag{2}$$

where x denotes the average anomaly score of $HNAN(V_i)$. Here *iScore* indicates the anomalous degree of $AAN(V_i)$, and *xScore* reflects the normality of $NHAN(V_i)$. We count the redundant $AS(V_i)$ and $Degree(V_i)$ only once. For example, in Fig. 2, *iScore*(*A*), *x*(*A*), and *xScore*(*A*) are 1.5, 1.67, and 1, respectively. Then we define *ixScore*(V_i) as:

$$ixScore(V_i) = iScore(V_i) + xScore(V_i). \quad (3)$$

Clearly, *ixScore*(V_i) is used to combine the multiple pieces of evidence with node V_i itself and its neighbors. If most neighbors of node V_i are anomalous and most neighbors of its *AAN*(V_i) are normal, node V_i is more likely to be the root cause.

In addition, as presented in [8], the resource utilization of host node h_i and the response time of deployed microservices (e.g., V_i) on h_i are correlated. For simplicity, we calculate the correlation between the response time metrics of V_{fe} ($|M|_{fe}$) and system utilization metrics of h_i ($|M|_i$) as follows:

$$Corr(V_{fe}, h_i) = \frac{\sum_{t=0}^{T}\left(|M|_{fe} - \overline{|M|}_{fe}\right)\left(|M|_{fe} - \overline{|M|}_i\right)}{\sqrt{\sum_{t=0}^{T}\left(|M|_{fe} - \overline{|M|}_{fe}\right)^2}\sqrt{\sum_{t=0}^{T}\left(|M|_{fe} - \overline{|M|}_i\right)^2}}. \quad (4)$$

This correlation function is the Pearson correlation coefficient between the metrics of V_{fe} and h_i. The value falls in [0,1]. In normal cases, the correlation between V_{fe} and h_i is closer to 0. Besides, the system utilization of h_i such as CPU, memory, I/O, and network utilization are ranked as the second phase ranking. The max value of *Corr*(V_{fe}, h_i) indicates the key anomalous metric. Finally, the anomaly weight *w* of V_i can be updated as:

$$w(V_i) = ixScore(V_i) \times \max Corr(V_{fe}, h_i). \quad (5)$$

Each time an anomaly is detected based on real-time metrics, the anomaly weight for each microservice in the SDG is recalculated for automatically updating. As shown in Fig. 3, the composition of *w* is the final anomaly weights *W* in the SDG. Then we normalize *W* for the random walk algorithm.

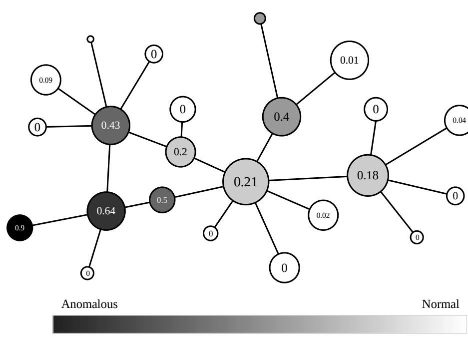

Figure 3. Example of anomaly weights in the SDG

F. Root Causes Ranking

Some methods [5, 23] rank the anomalies by the nodes or traces similarity. However, the microservices on root cause embedded request trace would be treated as anomalous with these methods. Moreover, the updated *ixScore* is based on its neighbors, and it is limited in a small range. To solve the above problems, we surfer from the whole SDG for further ranking the anomalies with the Personalized PageRank (PPR) algorithm, which performs well in the previous works [8, 20]. In the PPR algorithm, we use Personalized PageRank vector *v* to represent the anomaly weight in the SDG. And we define the transition probability matrix as *P*. Those nodes with a higher *AS* would have a higher access probability.

With PPR, we get the ranking list of root causes as the first phase ranking. Then we associate the root causes with the anomalous metrics ranking (the second phase) to get a two-phase ranking list, which helps developers mitigate the microservice failures, as shown in Fig. 1(e).

V. Experiments

In this section, we conducted experiments to compare our method with several state-of-the-art techniques. The experiments were designed to answer three research questions:

- **RQ1**: Does the proposed method outperform the state-of-the-art approaches in terms of different anomaly cases?
- **RQ2**: Is our approach effective enough to locate the root cause with fast speed?
- **RQ3**: Can our approach adapt to large-scale systems?

A. Setup

1) Experiment Settings. We evaluated the prototype of AAMR on two physical servers. Each physical server has an 8-core 2.40GHz CPU, 16GB of RAM, and Ubuntu 16.04 OS. And we installed Kubernetes 11.3.1, Istio[1] 1.4.5, Node Exporter[2] 1.41, and Prometheus[3] 6.3 on these servers for environment configuration. We used one server to run our system and another server to simulate the workload.

2) Benchmark. The benchmark of experiments is an online shop microservice system named Online-boutique[4], which contains 11 microservices. Particularly, since three microservices are mocked and a microservice is used for load generation, effects on these microservices are rather low, and we deployed them on the Kubernetes clusters but excluded them from the evaluation.

TABLE II. WORKLOAD GENERATION DETAIL

MS	cart	payment	currency	checkout	catalog	frontend	recommendation
users	100	100	100	100	100	100	100
rate(/s)	30	10	20	10	100	10	20

3) Data Collection. The workload was generated by Locust[5], a distributed load testing tool that simulates concurrent users in an application. Considering real user scenarios, we simulate different request rates for different microservices as shown in Table II. For system-level metrics, we used Node Exporter to collect CPU, memory, I/O, and network utilization metrics. And we used Prometheus, an open-source monitoring tool, to collect response time metrics. These metrics are collected at five-second intervals, and *T* is set as 150 seconds.

4) Fault Injection. To simulate real-world performance issues, we injected the following three types of failures: (*i*) *Latency Delay*. We used the feature of Istio to add a virtual service to instances, which has the effect of increasing the

[1] https://istio.io
[2] https://github.com/prometheus/node_exporter
[3] https://prometheus.io
[4] https://github.com/GoogleCloudPlatform/microservices-demo
[5] https://locust.io

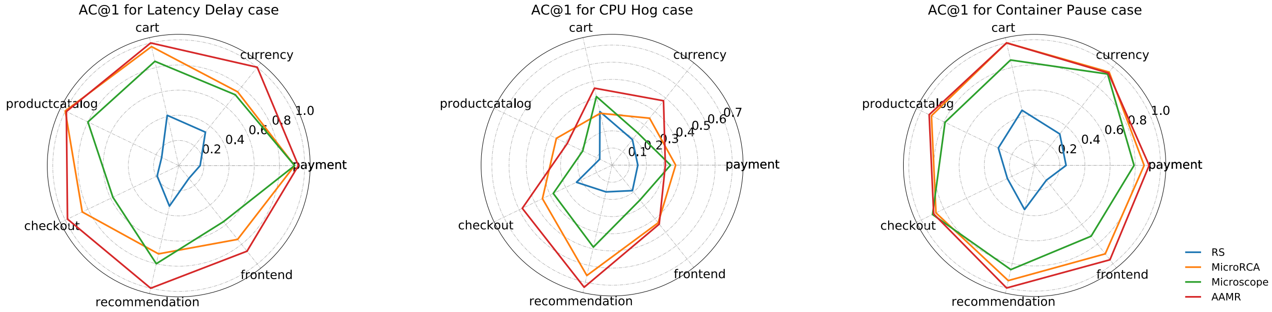

Figure 4. Performances of RS, MicroRCA, Microscope, and AAMR on different microservices

response time of a specified instance to 300ms. (*ii*) *CPU Hog*. The performance issue may be caused by the insufficient CPU allocated to the host. We used stress-ng[1] to stress the system CPU to 99% usage. As for container CPU usage, we limited the utilization of the injected instance by setting Kubernetes configurations. (*iii*) *Container Pause*: The "docker pause" command triggers a pause operation on the specified container. The container cannot be shut down directly because of the protection mechanism of Kubernetes.

5) Evaluation Metrics. To quantify the performance of each algorithm, we adopt the same evaluation metrics defined in [6]:

- Accuracy at top *k* (*AC@k*) indicates the probability that the top k on the ranking list hits the real root cause for all given anomaly cases. A higher *AC@k* score represents the algorithm identifying the root cause more accurately. In experiments, we choose k=1 and 3. Let $R[i]$ be the rank of each cause and V_{rc} be the set of root causes. *AC@k* is defined on a set of anomalies *A* as:

$$AC@k = \frac{1}{A}\sum_{a\in A}\frac{\sum_{i<k}(R[i]\in V_{rc})}{(\min(k,|V_{rc}|))} \tag{6}$$

- Average accuracy at top *k* (*Avg@k*) quantifies the overall performance of an algorithm, where *n* is the number of microservices. It is defined as:

$$Avg@k = \frac{1}{A}\sum_{a\in A}\sum_{1\le k\le n} AC@k \tag{7}$$

6) Baseline Methods. To evaluate the performance of AAMR, we compared it to the following baseline methods:

- **Random Selection (RS)**: Random selection randomly selects the possible anomalous microservices among all nodes without any domain knowledge.
- **Microscope**: Microscope [5] is a graph-based method to locate root causes. For Microscope implementation, we used the 3-sigma principle to detect anomalies and then added these anomalies into a candidate group. We collected the response time for calculating the similarity and ranking the anomalies in the candidate group.
- **MicroRCA**: MicroRCA [8] extracts an anomalous subgraph based on the SDG. For root cause localization, MicroRCA uses a Personalized PageRank algorithm, which is extended in our approach. To implement MicroRCA, we clustered the RT_i of microservices to extract the subgraph of anomalous nodes.

[1] https://kernel.ubuntu.com/cking/stress-ng

B. *RQ1: Performance Comparison*

We tested the performance of AAMR for different fault injection cases. Table III shows the performance in terms of *AC@*1, *AC@*3, and *Avg@*3 for all methods. We can observe that AAMR outperforms the baseline methods in most cases. In 10-round experiments, AAMR achieves an accuracy of 91% for *AC@*1 and 94% for *Avg@*3 on average, which outperforms the state-of-the-art methods. The result shows that AAMR gets 3.2% and 9.0% improvement than MicroRCA and Microscope for *AC@*3, respectively. It is also noticed that the experimental result of the CPU hog case is not as good as other cases because only computation-sensitive microservices are affected in the CPU hog case, e.g., the checkout service and recommendation service in Online-boutique.

TABLE III. PERFORMANCE COMPARISON

Metric	RS	MicroRCA	Microscope	AAMR	Improvement to MicroRCA	Improvement to Microscope
Overall						
*AC@*1	24%	90%	85%	**91%**	+1.1%	+7.0%
*AC@*3	38%	94%	89%	**97%**	+3.2%	+9.0%
*Avg@*3	31%	92%	90%	**94%**	+2.2%	+4.4%
Latency Delay						
*AC@*1	22%	92%	87%	**94%**	+2.2%	+8.0%
*AC@*3	43%	95%	90%	**97%**	+2.1%	+7.6%
*Avg@*3	37%	92%	90%	**95%**	+3.3%	+5.5%
CPU Hog						
*AC@*1	25%	**49%**	39%	48%	-2.0%	+23.1%
*AC@*3	36%	68%	59%	**70%**	+2.9%	+18.6%
*Avg@*3	35%	69%	61%	**70%**	+1.5%	+14.7%
Container Pause						
*AC@*1	33%	92%	90%	**95%**	+3.3%	+5.6%
*AC@*3	37%	**100%**	98%	**100%**	0%	+2.0%
*Avg@*3	41%	97%	94%	**98%**	+1.0%	+4.3%

In Fig. 4, we compared the performance of each method on different microservices. The result shows that AAMR outperforms other methods in most fault injection cases. MicroRCA performs better in some CPU hog cases because it calculates the correlation between the anomalous node and the host node, which is more accurate but has a higher overhead. However, AAMR performs better on average.

C. RQ2: Localization Time Comparison

Besides accuracy, developers expect to locate anomalies quickly. We set all methods running continuously, and only the top 1 ranking hits the root cause three times consecutively is considered successful. Table IV shows that the execution time of locating the root cause varies from methods, and AAMR takes less time to locate the root cause, i.e., 78% and 72% faster than Microscope and MicroRCA. Here the RS method is excluded in the comparison because of low accuracy.

TABLE IV. LOCALIZATION TIME COMPARISON

MS	cart	payment	currency	checkout	catalog	frontend	reco.	Avg
MicroRCA	15.2s	28.1s	33.5s	12.8s	9.4s	9.7s	26.3s	19.3s
MicroScope	6.7s	39.4s	43.5s	8.8s	29.4s	11.9s	32.5s	24.6s
AAMR	**3.3s**	**2.1s**	**2.0s**	**7.3s**	**2.1s**	**6.4s**	**14.2s**	**5.4s**

D. RQ3: Scalability Comparison

Scalability is the main feature of microservice systems. It is noticed that scaling out service replicas will increase the size of the SDG and make it more complicated to locate the root cause. We evaluated the impact of scaling out replicas from 1 to 10 for each microservice in Online-boutique. Fig. 5 shows that AAMR consistently maintains an accuracy of 82-91% for *AC@*1, which is higher than the state-of-the-art methods.

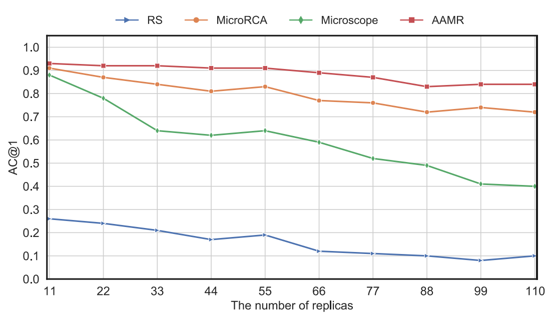

Figure 5. Comparison of scalability

VI. CONCLUSION AND FUTURE WORK

In this paper, we design a root cause analysis approach named AAMR. We extend the mRank algorithm to measure the anomaly weight of a node based on its adjacent nodes. After detecting the anomalies by a simple but effective clustering method, we give a two-phase ranking, which helps developers quickly diagnose the system failures. Experiments show that AAMR has an accuracy of 91% and an average accuracy of 94%, which outperforms the state-of-the-art methods.

In the future, we plan to cover more anomaly patterns by adding more metric types. Besides, we will try injecting more faults to test the performance of AAMR in case that multiple anomalies occur at the same time.

VII. ACKNOWLEDGMENT

This work is supported by the National Key R&D Program of China (No. 2018YFB1402800) and the National Natural Science Foundation of China (Nos. 62032016 and 61832014).

REFERENCES

[1] S. Newman, *Building Microservices*, O'Reilly Media, Inc, 2015.

[2] M. Cinque *et al*, "Microservices Monitoring with Event Logs and Black Box Execution Tracing," *IEEE Trans. Serv. Comput.*, pp. 1–1, 2019.

[3] J. Thalheim *et al.*, "Sieve: actionable insights from monitored metrics in distributed systems," in *IMC*, Las Vegas Nevada, Dec. 2017.

[4] K. Myunghwan, S. Roshan, and S. Sam, "Root Cause Detection in a Service-Oriented Architecture," *SIGMETRICS*, 2013.

[5] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments," *ICSOC*, p. 18, 2018.

[6] P. Wang *et al.*, "CloudRanger: Root Cause Identification for Cloud Native Systems," in *CCGRID*, Washington, DC, USA, May 2018, pp. 492–502.

[7] M. Ma and W. Lin, "MS-Rank: Multi-Metric and Self-Adaptive Root Cause Diagnosis for Microservice Applications," *ICWS*, 2019.

[8] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "MicroRCA: Root Cause Localization of Performance Issues in Microservices," in *NOMS*, Budapest, Hungary, Apr. 2020, pp. 1–9.

[9] Y. Ge, G. Jiang, M. Ding, and H. Xiong, "Ranking Metric Anomaly in Invariant Networks," *ACM Trans. Knowl. Discov. Data (TKDD)*, vol. 8, no. 2, pp. 1–30, Jun. 2014.

[10] S.-P. Ma, C.-Y. Fan, Y. Chuang, W.-T. Lee, S.-J. Lee, and N.-L. Hsueh, "Using Service Dependency Graph to Analyze and Test Microservices," in *COMPSAC*, Tokyo, Japan, Jul. 2018, pp. 81–86.

[11] B. H. Sigelman *et al.*, "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure," *GTR*, p. 14, 2010.

[12] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: problem determination in large, dynamic Internet services," in *ICDSN*, Washington, DC, USA, 2002, pp. 595–604.

[13] Z. Cai, W. Li, W. Zhu, L. Liu, and B. Yang, "A Real-Time Trace-Level Root-Cause Diagnosis System in Alibaba Datacenters," *Access*, vol. 7, p. 11, 2019.

[14] P. Liu *et al.*, "FluxRank: A Widely-Deployable Framework to Automatically Localizing Root Cause Machines for Software Service Failure Mitigation," in *ISSRE*, Berlin, Germany, Oct. 2019, pp. 35–46.

[15] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles - SOSP '09*, Big Sky, Montana, USA, 2009, p. 117.

[16] T. Jia, P. Chen, L. Yang, Y. Li, F. Meng, and J. Xu, "An Approach for Anomaly Diagnosis Based on Hybrid Graph Model with Logs for Distributed Services," in *2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, HI, USA, Jun. 2017, pp. 25–32.

[17] 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," *Journal of Systems and Software (JSS)*, vol. 159, p. 110432, Jan. 2020.

[18] Q. Du, T. Xie, and Y. He, "Anomaly Detection and Diagnosis for Container-Based Microservices with Performance Monitoring," *ICA3PP*, p. 13, 2018.

[19] H. Wang *et al.*, "GRANO: interactive graph-based root cause analysis for cloud-native distributed data platform," *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 1942–1945, Aug. 2019.

[20] Y. Meng *et al.*, "Localizing Failure Root Causes in a Microservice through Causality Inference," *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 2020, pp. 1-10.

[21] J. Weng, J. H. Wang, J. Yang, and Y. Yang, "Root Cause Analysis of Anomalies of Multitier Services in Public Clouds," *TON*, p. 14, 2018.

[22] A. Gulenko, F. Schmidt, A. Acker, M. Wallschlager, O. Kao, and F. Liu, "Detecting Anomalous Behavior of Black-Box Services Modeled with Distance-Based Online Clustering," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 912–915.

[23] L. Meng, F. Ji, Y. Sun, and T. Wang, "Detecting anomalies in microservices with execution trace comparison," *FGCS*, vol. 116, pp. 291–301, Mar. 2021.