ImpactTracer: Root Cause Localization in Microservices Based on Fault Propagation Modeling

Ru Xie*†, Jing Yang*‡, Jingying Li*†, Liming Wang*†
*Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
†School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{xieru,yangjing,lijingying,wangliming}@iie.ac.cn
‡Corresponding author

***Abstract*—Microservice architecture is embraced by a growing number of enterprises due to the benefits of modularity and flexibility. However, being composed of numerous interdependent microservices, it is prone to cascading failures and afflicted by the arising problem of troubleshooting, which entails arduous efforts to identify the root cause node and ensure service availability. Previous works use call graph to characterize causality relationships of microservices but not completely or comprehensively, leading to an insufficient search of potential root cause nodes and consequently poor accuracy in culprit localization.**

In this paper, we propose *ImpactTracer* to address the above problems. *ImpactTracer* builds impact graph to provide a complete view of fault propagation in microservices and uses a novel backward tracing algorithm that exhaustively traverses the impact graph to identify the root cause node accurately. Extensive experiments on a real-world dataset demonstrate that *ImpactTracer* is effective in identifying the root cause node and outperforms the state-of-the-art methods by at least 72%, significantly facilitating troubleshooting in microservices.

***Index Terms*—microservice, cloud-native, dependability, fault modeling, root cause identification**

I. Introduction

Microservice is gaining popularity as cloud computing matures increasingly and cloud-native gets embraced by the IT industry [1] [2]. Compared to monolithic applications, a microservice application boasts flexibility, scalability and agility because it dismantles service into a collection of microservices that can be developed and updated independently but interact and share data via network to provide service as a whole [3].

However, all of the benefits come at the cost of a surge in the number of microservices and an explosion of network communications, producing a complex system and a formidable troubleshooting task [4] [5]. In particular, mass and intricate interaction of microservices allows a single fault to spread out, cause cascading failures, and even render the whole service unavailable, underscoring the urgent need of locating the root cause node rapidly to ensure service reliability [6].

A number of methods have been proposed to identify the root cause node in microservices. They use call graph to characterize causality relationships of microservices and traverse it to trace the root cause node that best explains observed anomalies [7-11]. Unfortunately, their methods neither extract complete causalities nor assess them comprehensively, which further skews estimate of node "culpability", misleads root cause tracing, and ultimately results in poor accuracy.

In this paper, we aim to comprehensively evaluate node contribution to the detected anomalies so as to precisely locate the root cause node. We name our method as ***ImpactTracer*** and give an overview of it in Fig.1.

ImpactTracer starts with anomaly detection that monitors business golden indicators of microservices and uses Isolation Forest [12] to identify outliers. Triggered by detected anomalies, *ImpactTracer* proceeds to locate the culprit. To begin with, it constructs impact graph to model fault propagation in microservices, which is built by analyzing microservice interaction during the period of fault spread and presents all probable fault propagation paths. To comprehensively evaluate the odds of fault propagation from one node to another, *ImpactTracer* incorporates multiple critical factors and produces a single probability value. After that, starting from anomalous nodes, *ImpactTracer* exhaustively traverses the impact graph along reversed propagation paths to comprehensively evaluate node culpability. Each node is given a suspicion score that accumulates during that process. Finally, node with the highest score is regarded as the culprit of the observed anomalies, or the root cause node.

We conduct extensive experiments on a real-world dataset to evaluate the effectiveness of our method. The results demonstrate that *ImpactTracer* is superior in locating the culprit microservice and improves accuracy by at least 72.73% compared with the start-of-the-out methods.

In summary, we make the following contributions:

- We model fault propagation in microservices by giving a complete view of possible propagation paths and comprehensively assessing the corresponding probabilities.
- We design a backward tracing algorithm that works on the well-constructed model to accurately evaluate node culpability and precisely locate the root cause node.
- We extensively evaluate our method and prove its effectiveness and superiority in anomaly detection and root cause identification.

Organization. The rest of this paper is organized as follows. Section II introduces the background and our motivation. Section III presents our approach of anomaly detection. In Section IV, we elaborate our method of modeling fault propagation, followed by a detailed description of the backward tracing algorithm in Section V. Evaluation of *ImpactTracer* is shown in Section VI. Finally, we conclude our paper in Section VII.

978-3-9819263-7-8/DATE23/© 2023 EDAA

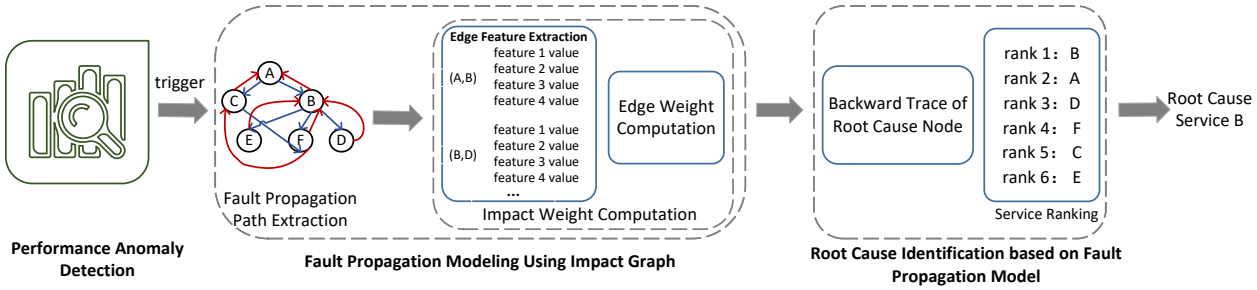


Fig. 1. Overview of *ImpactTracer*

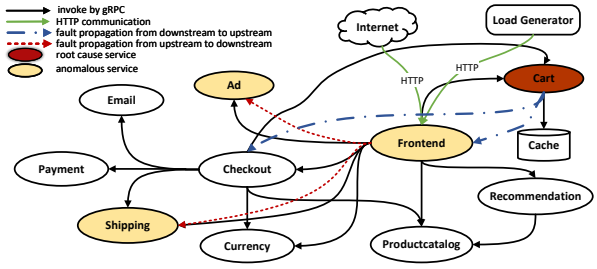


Fig. 2. Service architecture of Online Boutique

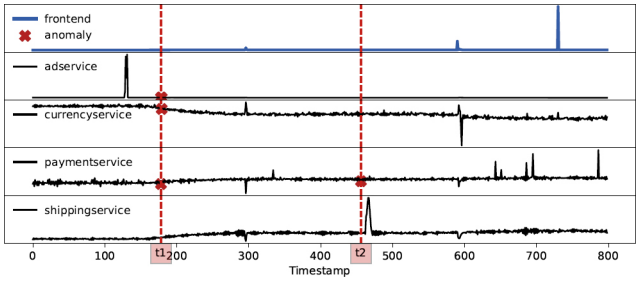


Fig. 3. Mean response time of microservices over time

II. Background and Motivation

In this section, we introduce the microservice architecture and the root cause localization problem. Previous methods and their limitations are presented to better clarify our motivation.

A. Microservice Architecture

In a microservice-based application, service is decomposed of multiple microservices that function independently while communicating with each other to provide service to external users. To coordinate all microservices, a central component *frontend* is developed to receive user requests and return response messages. Fig.2 shows architecture of a microservice application Online Boutique. To handle an online request, *frontend* invokes *checkout*, *currency*, etc. that further call other microservices. We dub invoking and invoked microservices as upstream and downstream microservices, respectively.

B. Root Cause Identification

Due to frequent interaction of microservices, a fault occurring at one microservice can spread out and result in performance degradation of others. Fig.3 shows mean response time of a subset of microservices in Online Boutique. At timestamp t_1, $fault$ 1 occurs at *adservice*. Shortly, multiple other microservices observe a prolonged response time. In order to resume normal operation and ensure service availability, developers need to locate and restore the culprit microservice that causes widespread anomalies as soon as possible.

C. Challenges and Proposed Methods

It is nontrivial to conduct fault analysis in a system as large and dynamic as a microservice application. Moreover, intricate interaction of microservices makes it formidable to untangle fault spread and locate the root cause node.

To facilitate root cause localization in microservices, previous works [7-11] adopt call graph to characterize interaction of microservices, in which an edge $u \rightarrow v$ represents u calls v and therefore can be affected by v. Then, they traverse the call graph along reversed edges to trace the culprit of the detected anomalies. Various traverse algorithms are adopted by different methods. MonitorRank [7], CloudRanger [8] and AutoMAP [9] use random walk algorithms [13] in which the transition probability from node u to v depends on metric correlation of microservice v with that of anomalous nodes. In other words, they assume nodes more correlated with anomalies are more likely to be the culprit. MicroHECL [10] traverses edges from anomalous nodes in the opposite direction to search possible root cause nodes, which are further ranked on metric similarity to the initial anomalous nodes. Microscope [11] takes a more straightforward strategy that simply ranks anomalous microservices according to metric similarity.

D. Motivations

Despite mitigating the troubleshooting problem, previous works obtain poor accuracy due to the following limitations:

Limitation 1: Neglect essential fault propagation paths. Tracing the root cause node in the "called-call microservice" direction, previous works assume only downstream microservices can affect upstream ones, overlooking the fact that faults can also spread from upstream to downstream. In Fig.2, four anomalous nodes are detected with *cart* being the root cause. When a network failure ($fault$ 3) occurs at *cart*, the upstream *frontend* is affected and becomes anomalous. Then, through invocations of $frontend \rightarrow ad$ and $frontend \rightarrow shipping$, *ad* and *shipping* are impacted. If we erase fault propagation paths from upstream to downstream, we cannot explain anomalies at *ad* or *shipping*, nor can we find the real culprit.

Limitation 2: Not comprehensively assess fault propagation probabilities. When searching for potential culprits, previous works use metric similarity or correlation to measure the probability of fault spreading from one node to another. However, such a method overlooks other factors that contribute

TABLE I
BUSINESS GOLDEN INDICATORS USED IN ANOMALY DETECTION

Indicators	***Definitions***
Response rate	Pct. of successfully received responses
Success rate	Pct. of successfully handled requests
Mean response time	Average time it takes to handle a request
Transaction count	Total number of handled requests

and consequently misleads culprit tracing. For example, in Fig.3, at t_2, an anomay is detected at *payment*, with which *frontend* has the most similar metric change pattern. However, the real culprit is *shipping*, the least similar node to *payment*.

The above limitations combine to mislead culprit tracing and result in poor accuracy in root cause localization. In this paper, we give a complete view of fault propagation paths (§IV-A) and comprehensively evaluate their probabilities (§IV-B), facilitating accurate culprit identification (§V).

III. PERFORMANCE ANOMALY DETECTION

In highly dynamic microservice environments, service performance has peaks and troughs even when no anomaly occurs, making it challenging to distinguish real anomalies from normal fluctuations. Considering that, we monitor multiple business golden indicators (BGI) of microservices and adopt decision tree-based Isolation Forest [12] instead of threshold-based 3-sigma rule used in previous works to achieve high accuracy and great adaptability in anomaly detection.

Business golden indicators are an effective way to monitor health status of a microservice and spot problems. The indicators we use are shown in Table I. Previous works [7] [9] monitor and conduct anomaly detection on *frontend* alone because of its role as the communication hub of microservices and a checkpoint for service operation status. However, our analysis on a real-world dataset demonstrates that not all anomalies affect *frontend*. For instance, in Fig.3, when $fault$ 2 occurs at t_2, *frontend* performs steadily and no anomaly occurs. In case of missing any faults, we monitor all microservices and apply Isolation Forest on collected data of each one.

When a series of outliers are detected in any microservice, we determine that a performance anomaly has occurred, caused by a fault somewhere in the service. Furthermore, for each outlier a, an anomaly score (AS) is calculated as Equation (1) to measure its deviation from the normal level.

$$AS(a) = BGIValue(a) - \frac{\sum_{t \in T} BGIValue(d_t)}{length(T)} \quad (1)$$

where $BGIValue(d_t)$ represents the normalized value of business golden indicators of a microservice at timestamp t while T represents the period of time T before a occurs.

IV. FAULT PROPAGATION MODELING

When a performance anomaly is detected, *ImpactTracer* proceeds to locate the culprit microservice where the fault occurs initially and advances to compromise other nodes. In this section, we construct an impact graph to provide a complete view of possible fault propagation paths and a comprehensive assessment of their probabilities.

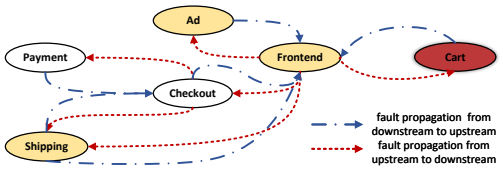


Fig. 4. Impact graph of fault 3

A. Extract Fault Propagation Paths

To handle a request, an upstream microservice u calls downstream v that returns results, during which u and v affect each other. On the one hand, faults at u, such as network latency, can reduce the number of requests oriented to v, causing anomalous transaction count. On the other hand, a CPU or memory fault at v can result in prolonged response time and lower success rate of both v and u. This means that faults can spread from upstream to downstream and vice versa.

To capture all possible fault propagation paths, we collect and analyze microservice invocations over a period of time before the anomaly is detected, during which a fault occurs and spreads across the service. We define this period as the fault propagation time window (***fpw***) and adjust it according to demand. We record the calling and called microservices of each collected invocation and integrate all of them. Finally, for each microservice, we obtain the number of calls initiated and received by it, denoted as *numCalling* and *numCalled* respectively. In addition, for a specific invocation $u \rightarrow v$, we mark the number of times it occurred in *fpw* as *count(u,v)*.

With all the above information, we build a skeleton of the impact graph to characterize interaction of microservices and present all possible fault propagation paths during *fpw*. In the impact graph, each microservice is represented as a specific node and invocation $u \rightarrow v$ corresponds to both edge (u, v) and (v, u) that represent the fault propagation paths from u to v and v to u. Fig.4 shows the impact graph of $fault$ 3 with *fpw* set to 13.2 s. Compared with Fig.2, Fig.4 does not contain edges from $checkout$ to $cart$ or vice versa because the corresponding invocation is not collected during *fpw*.

B. Compute Fault Propagation Probability

In the impact graph, an edge (u, v) represents impact of u on v and embodies a fault propagation path from u to v. Of all the nodes that might have an effect on v, various properties of nodes produce varied strengths of impacts and consequently disparate fault propagation probabilities, reflected on distinct weights of edges in the impact graph. We introduce factors that contribute to different impact intensities and our method of calculating fault propagation probabilities.

Feature Extraction. For each edge (u,v), we extract four features to characterize to what extent u can impact v.

- *Node Anomaly Degree.* Closely interconnected, v is subject to performance anomalies at u. Usually, it is resilient and can adapt to external disturbances, but only if the impact is within the scope of self-adaptation. Therefore, the more anomalous u is and the longer its impact persists, the more likely v is affected. We define *ADegree(u)* to measure the anomaly degree of u and calculate it as follows:

$$ADegree(u) = \begin{cases} \max\limits_{t \in fpw} AS(d_t) & \forall t \in fpw, AS(d_t) > 0 \\ s \times \max\limits_{t \in fpw} AS(d_t) & \exists t \in fpw, AS(d_t) = 0 \end{cases} \tag{2}$$

where *s* is adjustable and ranges from 0 to 1. Nodes exhibit abnormalities throughout *fpw* are highlighted because this demonstrates an intense fault with profound impact.

- *Node Activity Degree.* An active node interacts frequently with others, acting as a bridge in information transmission and an agent in fault propagation, which makes it more likely to impact others. We measure activity of node u with the probability that node u appears in the shortest paths between any other two nodes, as shown as Equation(3).

$$Activity(u) = \sum_{p \neq u \neq q} \frac{\sigma(u)}{\sigma_{pq}} \tag{3}$$

$\sigma(pq)$ and $\sigma(u)$ represent the total number of all-pair shortest paths and that of those passing through node u, respectively.

- *Intimacy.* Intuitively, the more frequently u and v communicate, the more likely u will affect v. We define *Intimacy(u,v)* as follows to measure the closeness between u and v:

$$Intimacy(u, v) = \frac{count(u, v)}{numCalled(v)} \tag{4}$$

It represents the proportion of invocation $u \rightarrow v$ in communications of v, measuring the importance of u to v.

- *Similarity.* According to previous works [7] [9] [11], if the performance metrics of two microservices have similar change pattern, it is probable that they affect each other. We follow this philosophy and define $Similarity(u, v)$ as:

$$Similarity(u, v) = \frac{cov(BGI(u), BGI(v))}{\sigma_{BGI(u)} \sigma_{BGI(v)}} \tag{5}$$

It is derived from *Pearson Correlation Coefficient* [14] and measures correlation between business golden indicators of u and v. $Similarity(u, v)$ ranges between -1 and 1, with a greater absolute value indicating higher similarity.

Probability Computation. For edge (u,v), the above four factors combine to determine impact of u on v and the probability that the fault spreads from u to v. To incorporate all the features to compute the final probability, we resort to *Principal Component Analysis* (PCA) [15] which is widely used in dimensionality reduction [16].

PCA "compresses" edge attributes contained in four feature values nearly losslessly to a single weight value $w_{(u,v)}$ that represents the probability of fault propagation from u to v. A large $w_{(u,v)}$ indicates a high probability.

V. Backward Trace of Root Cause Node

Impact graph characterizes interaction of microservices, modeling fault propagation throughout the service. Reverse edges in the impact graph, we obtain a reversed impact graph (*RIG*) in which an edge (u,v) with weight $w_{(u,v)}$ represents that u is impacted by v, and specifically that anomaly at u is propagated from v with probability of $w_{(u,v)}$.

Intuitively, for an anomalous node, if we recursively search for nodes that might cause anomaly at it along edges of *RIG*, we can locate the culprit of it. Our backward tracing algorithm starts from anomalous nodes and exhaustively traverses *RIG* to trace the root cause node that is most likely to be responsible for the detected anomalies.

Suspicion Score. The basic idea of our approach is to give each node a suspicion score (*SS*) that accumulates through the traverse of *RIG* to estimate the probability of being the root cause node. A large *SS* means high probability.

The key insights are that for node v: **1)** the more links v receives in *RIG*, the more likely it is to contribute to the detected anomalies, therefore, the more likely v is the root cause node. **2)** links from nodes with high *SS* to v in *RIG* add up suspicion of v because it might disseminate fault to those nodes. **3)** weight of an edge linking to v also matters because it represents the likelihood that v is to blame. Therefore, suspicion score of v can be calculated as Equation(6).

$$SS(v) = \sum_{(u,v) \in RIG.edges} SS(u) \times w^*_{(u,v)} \tag{6}$$

where

$$w^*_{(u,v)} = \frac{w_{(u,v)}}{\sum\limits_{(u,p) \in RIG.edges} w_{(u,p)}} \tag{7}$$

Note that we normalize edge weights to ensure that, for each node, the sum of weights of its outgoing edge equals 1.

Comprehensive assessment of node suspicion. A fault occurs at a node and spreads in the service through fault propagation paths represented as edges of an impact graph, finally leading to detected anomalous nodes. To trace the root cause node, we simulate the reversed process of fault propagation by exhaustively traversing *RIG* and iteratively calculating Equation(6) for each node to comprehensively evaluate node suspicion. Matrix is used to accelerate computation.

For an application consisting of *n* microservices, the corresponding *RIG* contains *n* nodes. Let **M** be the transition matrix of *n* by *n* elements, each of which M_{vu} denotes the probability that v causes an anomaly at u and can be calculated by:

$$M_{vu} = \begin{cases} w^*_{(u,v)} & (u,v) \in RIG.edges \\ 0 & (u,v) \notin RIG.edges \end{cases} \tag{8}$$

We use vector **SSV** = $[ss_0, ss_1, \cdots, ss_{n-1}]$ to demonstrate suspicious scores of *n* nodes. Furthermore, since our backward trace starts from the detected anomalous nodes, we define a seed vector **Seed** = $[s_0, s_1, \cdots, s_{n-1}]$ where $s_i = 1$ for $i \in [0, n)$ if the corresponding node is anomalous.

Then, the *q*th iteration of computing suspicion scores of *n* nodes is defined as:

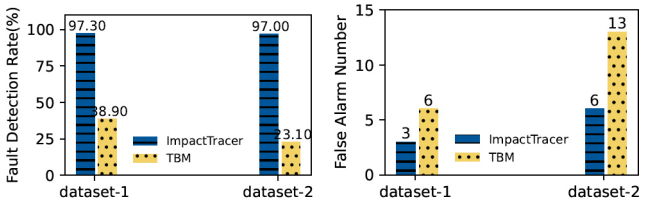
$$\boldsymbol{SSV^{(q)}} = \begin{cases} \boldsymbol{Seed \cdot M} & q = 1 \\ \boldsymbol{SSV^{(q-1)} M} & q > 1 \end{cases} \tag{9}$$

Root cause node Identification. We iteratively calculate Equation(9) until it reaches convergence to obtain the final suspicion scores of all nodes. Finally, all the nodes are ranked in descending order according to suspicion scores, with the one top of the list determined as the root cause node. Alg.1 illustrates the backward tracing process. It takes a reversed impact graph and a set of detected anomalous nodes as input and returns a list of candidate culprits.

Algorithm 1 Backward Tracing Root Cause Node

Input: Reverse Impact Graph G, anomalous nodes V^b, threshold δ;
Output: list of k candidate root cause nodes sorted by suspicion score in descending order

n = *count(G.nodes)*
M = $\boldsymbol{O_{n\times n}}$, $\boldsymbol{SSV}$ = $\boldsymbol{SSV_t}$ = **Seed** = $\boldsymbol{O_{1\times n}}$
for *(i,j)* $\in$ *G.edges* **do**
    $M_{ji} = w^*_{(i,j)}$
$\boldsymbol{SSV} = \boldsymbol{Seed} \cdot \boldsymbol{M}$
while $sum(|\boldsymbol{SSV} - \boldsymbol{SSV_t}|) > \delta$ **do**
    $\boldsymbol{SSV_t} = \boldsymbol{SSV}$
    $\boldsymbol{SSV} = \boldsymbol{SSV_t} \cdot \boldsymbol{M}$
$DescendingOrder(\boldsymbol{SSV})$
return list of i for SSV_i in $\boldsymbol{SSV}$



(a) Fault detection rate (b) False alarm number

Fig. 5. Results of Anomaly Detection

VI. Evaluation

In this section, we conduct extensive experiments to evaluate the effectiveness of *ImpactTracer*.

A. Dataset

The data we use comes from a real-world microservice application based on Hipster-Shop [17], a widely used microservice benchmark. Different groups of faults are injected into two identical testbeds and the generated data is collected respectively (dataset-1 and dataset-2), each of which contains over 20 million invocations of 10 microservices within 2 days.

B. Evaluation Metrics

To evaluate effectiveness of root cause identification, we use two metrics already employed in previous works [5] [9] [10].

Top-k Accuracy (A@k) refers to the probability that the root cause node is included at the top k of the result list. A great $A@k$ indicates an effective method.

Average Rank(AvgR) refers to the average rank of the root cause nodes in the result list for all faults. The smaller *AvgR* is, the more accurate the method is.

C. Effectiveness of anomaly detection

Fig.5 shows anomaly detection results of *ImpactTracer* and the previous threshold-based method (*TBM*) [7] [9]. It shows that *ImpactTracer* can detect faults with a higher fault detection rate (*FDR*) of over 97% and fewer false alarms, significantly improving fault detection accuracy.

Further analysis of the datesets demonstrates that only 48.6% of the injected faults (27 of 64) cause anomalies at *frontend*, making *TBM* miss most of them. Moreover, rapidly changing microservice environments further undermine effectiveness of the rigid threshold-based method. On the contrary, *ImpactTracer* monitors all microservices simultaneously and uses adaptable Isolation Forest to detect anomalies in business

TABLE II
Results of root cause node identification

Datasets	*Method*	*A@1*	*A@2*	*A@3*	*AvgR*
Dataset-1	**ImpactTracer**	**67.7%**	**83.3%**	**91.7%**	**1.57**
	MonitorRank	8.3%	16.7%	25%	5.16
	MicroScope	0	8.3%	8.3%	6.25
	ImpactTracer(-p)	8.3%	50%	75%	3.33
	ImpactTracer(s)	58.3%	66.7%	75%	2.50
Dataset-2	**ImpactTracer**	**55.6%**	**66.7%**	**88.9%**	**2.22**
	MonitorRank	11.1%	33.3%	55.6%	4.88
	MicroScope	0	22.2%	44.4%	5.66
	ImpactTracer(-p)	44.4%	55.6%	66.7%	3.11
	ImpactTracer(s)	44.4%	55.6%	55.6%	4.33

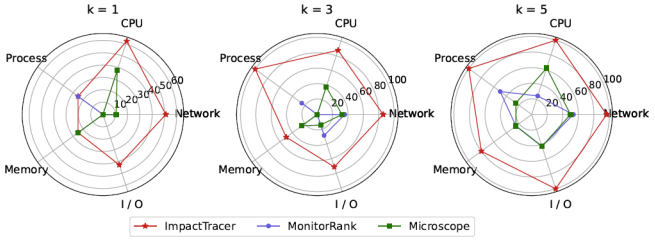


Fig. 6. A@k of different type of faults

golden indicators that are direct and potent demonstrations of service health condition, contributing to accurate detection.

D. Effectiveness of root cause microservice identification

We compare *ImpactTracer* with classic MonitorRank [7] and Microscope [11] and show the results in Table II. Furthermore, to assess the contribution of fault propagation modeling, we revise *RIG* by removing paths from upstream to downstream microservices or by solely using *similarity* to calculate fault spread probabilities. Results of Alg.1 on the revised *RIG* are listed as *ImpactTracer(-p)* and *ImpactTracer(s)*, respectively.

From Table II we can draw the following conclusions:

Conclusion 1: *ImpactTracer* can accurately locate the root cause node with high *Top-k Accuracy* and small *AvgR*. As illustrated in Table II, *ImpactTracer* achieves *A@1* of 67.7%, indicating that it is highly probable that the top of the returned list is the real culprit. *A@3* as high as 91.7% and 88.9% in the respective two datasets declare that we can find the root cause node just by analyzing the top 3 microservices of the returned list. Moreover, *ImpactTracer* achieves *AvgR* of 1.57 and 2.22 on two datasets, respectively, meaning that in the worst case, on average we only need to analyze 2.22 microservices before locating the real root cause node.

Conclusion 2: Fault propagation modeling plays a critical role in accurate root cause microservice localization. For a fault, *RIG* characterizes all possible fault propagation paths and their probabilities. If we erase edges from upstream microservices to downstream ones, the corresponding fault propagation paths are omitted, leading to incomplete search in the backward tracing algorithm. As we can see from Table II, compared with *ImpactTracer*, *ImpactTracer(-p)* decreases *A@1* by 87.74% and more than doubles *AvgR* in dataset-1. In other words, complete modeling of fault propagation paths increases overall *A@1*, *A@2* and *A@3* by 61.55%, 31.25% and 17.65%, respectively, significantly eliminating $limitation$ 1.

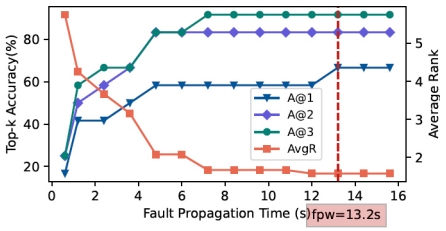


Fig. 7. Root cause effectiveness with different *fpw*

Previous works regard *similarity* as the only factor that determines fault spread. However, results in Table II underscore the importance of taking several factors into consideration. As shown in Table II, when *similarity* is only used to calculate fault propagation probabilities (see *ImpactTracer(s)*), *A@1*, *A@2* and *A@3* decrease by 15.38%, 18.75% and 17.65% on average, respectively. It highlights the importance of *anomaly degree*, *activity* and *intimacy* in fault propagation, as well as effectiveness of our method in mitigating *limitation* 2.

Conclusion 3: *ImpactTracer* significantly outperforms MonitorRank and MicroScope in locating root cause microservice. In dataset-1, MonitorRank yields *A@3* of 25% and *AvgR* of 5.16, failing to locate root causes of most faults. Microscope performs even worse. We attribute their poor performance to the neglect of critical fault propagation paths and the overlook of several critical factors that have an impact on fault spread. Specifically, as we can see from Table II, on dataset-2, *ImpactTracer(s)* improves *A@1* by 75% compared with MonitorRank, meaning we can improve the accuracy of root cause localization by 75% only by working on complete fault propagation paths. Similarly, results of *ImpactTracer(-p)* and Microscope demonstrate the contribution of a comprehensive assessment of fault propagation probabilities. In addition, Fig.6 shows *A@k* of the three methods for different types of faults. *ImpactTracer* performs equally well on different types of faults and outperforms the other two methods.

In conclusion, our method combines the benefits of complete modeling of fault propagation paths and an comprehensive evaluation of the corresponding probabilities. It outperforms MonitorRank and Microscope by at least 72.73% and exhibits a great superior in root cause localization.

E. Impact of Fault Propagation Window

When a fault occurs, we collect invocations during *fpw* and construct an impact graph to assist in root cause localization. So we evaluate the impact of *fpw* on the effectiveness of culprit identification. As illustrated in Fig.7, *A@k*(k = 1,2,3) increases and *AvgR* gradually decreases until *fpw* reaches 13.2s, when *ImpactTracer* achieves optimal performance.

The observations are reasonable because: **1)** longer *fpw* means more microservice interaction during fault spread is captured, making our model of fault propagation closer to the fact. **2)** as time passes, the impact of a fault recedes and fault propagation peters out, so invocations beyond this time interval (13.2s in our datasets) make no difference to the final result.

VII. Conclusion

In this paper, we propose *ImpactTracer* to solve the troubleshooting problem in microservices from the perspective of fault propagation modeling. *ImpactTracer* applies Isolation Forest on business golden indicators to detect anomalies, which is more accurate and adaptable than traditional methods. To identify the root cause microservice, *ImpactTracer* constructs an impact graph to fully model the paths and probabilities of fault propagation, laying the foundation for the backward tracing algorithm that exhaustively traverses the graph to trace the root cause node. Extensive experiments and the results on real-world data prove the effectiveness and superiority of *ImpactTracer* in culprit localization in microservices.

References

[1] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M. K. Reiter, and V. Sekar, "Gremlin: Systematic resilience testing of microservices," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016, pp. 57–66.

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

[3] N. Kratzke and P.-C. Quint, "Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study," *Journal of Systems and Software*, vol. 126, pp. 1–16, 2017.

[4] 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 Transactions on Software Engineering*, vol. 47, no. 2, pp. 243–260, 2018.

[5] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, and X. Li, "Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *Proceedings of the Web Conference 2021*, 2021, pp. 3087–3098.

[6] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: practical and scalable ml-driven performance debugging in microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 135–151.

[7] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 93–104, 2013.

[8] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 492–502.

[9] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "Automap: Diagnose your microservice-based web applications automatically," in *Proceedings of The Web Conference 2020*, 2020, pp. 246–258.

[10] D. Liu, C. He, X. Peng, F. Lin, C. Zhang, S. Gong, Z. Li, J. Ou, and Z. Wu, "Microhecl: High-efficient root cause localization in large-scale microservice systems," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2021, pp. 338–347.

[11] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *International Conference on Service-OrientedComputing*. Springer, 2018, pp. 3–20.

[12] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[13] G. F. Lawler and V. Limic, *Random walk: a modern introduction*. Cambridge University Press, 2010, vol. 123.

[14] H. Abe and S. Tsumoto, "Analyzing behavior of objective rule evaluation indices based on a correlation coefficient," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 2008, pp. 758–765.

[15] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.-R. Mullers, "Fisher discriminant analysis with kernels," in *Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop (cat. no. 98th8468)*. Ieee, 1999, pp. 41–48.

[16] J. Shlens, "A tutorial on principal component analysis," *arXiv preprint arXiv:1404.1100*, 2014.

[17] https://github.com/GoogleCloudPlatform/microservices-demo, 2022.