TraGraphRCA: Practical Multi-Level Root Cause Analysis for Microservice with Trace-Graph Fusion

HAIYU HUANG, Sun Yat-sen University, China
XIAOYU ZHANG, Huawei, China
PENGFEI CHEN, GUANGBA YU, and ZILONG HE, Sun Yat-sen University, China
QIUAI FU, Huawei, China
MICHAEL R. LYU, The Chinese University of Hong Kong, China

Root cause analysis (RCA) for large-scale microservice systems is a crucial and challenging problem. Service dependency graph and trace are two widely-used data sources in RCA. However, existing dependency-graph-based RCA approaches lack in-depth analysis of individual requests. On the other hand, trace-structure-based RCA approaches ignore anomalies across multiple traces. Moreover, most of existing RCA approaches fail to provide fine-grained analysis. In this study, we present *TraGraphRCA*, a practical multi-level microservice RCA approach that comprehensively combines the trace-structure-based analysis and graph-based analysis. *TraGraphRCA* constructs multi-level trace template patterns and service dependency graphs in an offline manner. During online analysis, *TraGraphRCA* utilizes both trace structure-status information and dependency graph information to locate the root cause service instance and the specific root cause event. Experimental results demonstrate that *TraGraphRCA* achieves a significantly higher average top-1 accuracy compared to seven baseline methods on two datasets. Moreover, *TraGraphRCA* has been deployed in a large real-world production system for 8 months and has been used to handle over 900 performance or reliability issues. It achieves an accuracy of over 80% in RCA, and the analysis time is always lower than 3 minutes.

CCS Concepts: • **Software and its engineering → Cloud computing**; • **General and reference → Reliability**; **Performance**.

Additional Key Words and Phrases: Root Cause Analysis, Multi-Level Diagnosis, Microservice

ACM Reference Format:
Haiyu Huang, Xiaoyu Zhang, Pengfei Chen, Guangba Yu, Zilong He, Qiuai Fu, and Michael R. Lyu. 2018. TraGraphRCA: Practical Multi-Level Root Cause Analysis for Microservice with Trace-Graph Fusion. *J. ACM* 37, 4, Article 111 (August 2018), 22 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Microservice architecture has become the mainstream framework for developing and building cloud-native applications. For industrial microservice applications, they typically consist of dozens to thousands of services with multiple instances [42, 48]. Running in a highly uncertain and dynamic environment, microservices inevitably suffer from reliability and performance issues [12, 45]. To

Authors' Contact Information: Haiyu Huang, huanghy95@mail2.sysu.edu.cn, Sun Yat-sen University, Guangzhou, China; Xiaoyu Zhang, zhangxiaoyu87@huawei.com, Huawei, Shenzhen, China; Pengfei Chen, chenpf7@mail.sysu.edu.cn; Guangba Yu, yugb5@mail2.sysu.edu.cn; Zilong He, hezlong@mail2.sysu.edu.cn, Sun Yat-sen University, Guangzhou, China; Qiuai Fu, fuqiuai@huawei.com, Huawei, Shenzhen, China; Michael R. Lyu, lyu@cse.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong, China.

ACM 1557-735X/2018/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

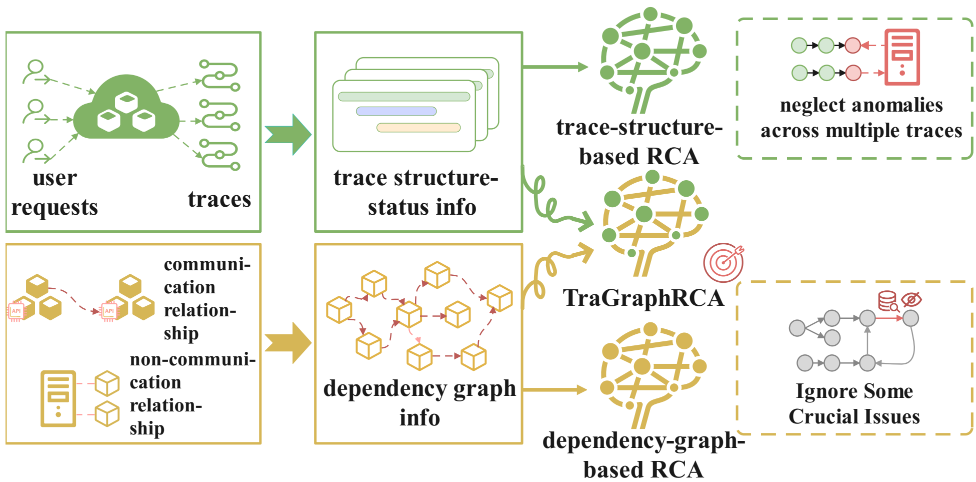


Fig. 1. Trace structure info and dependency graph info are combined to localize root causes in TraGraphRCA.

enable Site Reliability Engineers (SREs) to timely mitigate failures, it is desirable to automate root cause analysis (RCA) from thousands of services within complex dependencies [43].

Service dependency graph shows the dependencies between services in applications, providing rich context when assessing risks and understanding the system, which has been widely used by SREs to diagnose failures [7, 22, 39, 41]. Recent researches [7, 10, 22, 36–38] use dependency-graph-based approaches to locate root causes leveraging the abnormal edges between adjacent microservices within the graph. However, such methods lack in-depth analysis focused on individual requests, resulting in insufficient analysis, such as the ignorance of some crucial issues. As shown in Fig. 1, some issues which only affect specific requests escape from RCA.

On the other hand, with the support of specifications such as OpenTelemetry [24] and SkyWalking [34], distributed tracing [33] has been widely adopted in industrial microservice systems. Each trace records the end-to-end paths of requests across service instances and each operation (i.e., service invocation). Traces have been widely used in RCA for microservice systems. Existing trace-structure-based RCA approaches [20, 23, 29, 40, 45] locate the root causes by analyzing the differences in structure or state information between abnormal traces and normal traces. Nevertheless, due to the complex dependencies and fault propagation patterns in microservice systems, anomalies not only propagate within a single trace, but also across multiple traces [22], as shown in Fig. 1. Therefore, existing approaches which neglect anomalies across multiple traces obtain lower accuracy in practice (shown in § 4.2.1).

Moreover, existing dependency-graph-based and trace-structure-based RCA approaches are limited in the fine-grained analysis of root causes, as dependency graphs or traces provide rich information across service instances, but offer poor information within services. To obtain finer-grained information, events are inserted into trace logic to involve the information within service instances [26]. Therefore, incorporating events in RCA can help us obtain more detailed root cause reports, thus speeding up the fault mitigation.

***TraGraphRCA* Approach.** To pinpoint root causes of availability and performance issues in microservice systems, we propose a practical multi-level RCA approach called *TraGraphRCA*. The core idea of *TraGraphRCA* is to combine in-depth trace-structure-based analysis and overall dependency-graph-based analysis to localize root causes at multiple levels. It comprises two main phases, namely the construction phase and the diagnosis phase. In the construction phase, *TraGraphRCA* collects trace data periodically at normal state. It extracts and aggregates information from these trace data to build multi-level template patterns. Simultaneously, *TraGraphRCA* creates a service dependency graph by considering both communication and non-communication relationships among service instances. To enhance the efficiency of graph traversal, we store and maintain the graph in the form of bitmaps [14].

During the diagnosis phase, *Level-by-level Analyzer* (§ 3.2) retrieves trace data within the abnormal time window. It analyzes these trace data at service-span level and log-event level to mine the difference pairs between expected patterns and violated patterns. *Trace-Based Analyzer* (§ 3.3) then identifies suspicious spans based on the multi-level difference pairs. It also evaluates the significance of each suspicious span through trace-based analysis. Finally, *Graph-Based Analyzer* (§ 3.4) incorporates the results of trace-based analysis into the service dependency graph and applies PageRank [27] algorithm for graph-based analysis. It ultimately provides a list of potential root causes, which SREs can refer to for possible root cause service instances and specific root cause events.

To validate the effectiveness and efficiency of *TraGraphRCA*, we constructed two datasets, one from a large real-world production system and another from two widely-used microservices benchmarks namely TrainTicket [6] and OnlineBoutique [8]. Experimental results demonstrate that *TraGraphRCA* achieves significantly higher average top-1 accuracy (82.70%) compared to seven baseline methods at both service level and event level. On average, it outperforms them from 39.01% to 76.92%. In terms of practical application, *TraGraphRCA* has already been deployed in Huawei Cloud for 8 months. It has been used to handle over 900 performance or reliability issues with an accuracy of over 80%. SREs and developers have provided feedbacks that the analysis results from *TraGraphRCA* have effectively helped them improve their efficiency and save on manpower cost.

Contributions. This study makes the following contributions.

- We propose a multi-level trace analysis method at service-span level and log-event level, which not only identifies the root cause service instance but also provides specific root cause events.
- We propose a practical RCA approach called *TraGraphRCA*, which combines both trace-structure-based analysis and dependency-graph-based analysis to localize root causes. The implementation of *TraGraphRCA* is publicly available at [35].
- We constructed two datasets using 13 production microservice systems and 2 widely-used microservices benchmarks to validate *TraGraphRCA*. Experimental results demonstrate the effectiveness and efficiency of *TraGraphRCA*.
- We have deployed *TraGraphRCA* in Huawei Cloud for 8 months. During production operations, *TraGraphRCA* handles over 900 issues and achieves an accuracy of over 80% in RCA with an average analysis time less than 3 minutes.

2 Background and Motivation

2.1 Background

Trace information at service-span level. A trace comprises a series of service operations (spans) interconnected through context propagation, forming a tree-like topology [32]. The structure of a trace reflects which services involved in a request, as well as the order and hierarchy of their invocations. Additionally, the status information on each span, such as call duration and return status code, also provides crucial insights for subsequent analysis. However, the original trace information can only provide coarse-grained information at service operation level.

Trace information at log-event level. In many end-to-end tracing tools (e.g., OpenTelemetry [24]), spans include log events [26], which can be seen as structured log messages with timestamps and event information. These events represent a series of meaningful sub-operations that occur within a span. The finer-grained status information on events such as latency, execution parameters and results helps us conduct in-depth RCA.

Dependency graph information. In production microservice systems, there are complex dependencies between different microservices. These dependencies can be categorized into communication relationships and non-communication relationships [22]. Communication relationships can be obtained through network sockets or by aggregating multiple traces to build a call graph [41]. For systems using service mesh, the network components of the service mesh can also provide communication relationship information [3]. Non-communication relationships typically involve competition for a shared resource or concurrent access to configuration files [22]. This information can be obtained by examining the deployment configuration of services and nodes. These dependencies between services are usually stored and maintained in the form of a service dependency graph, which is helpful for conducting RCA.

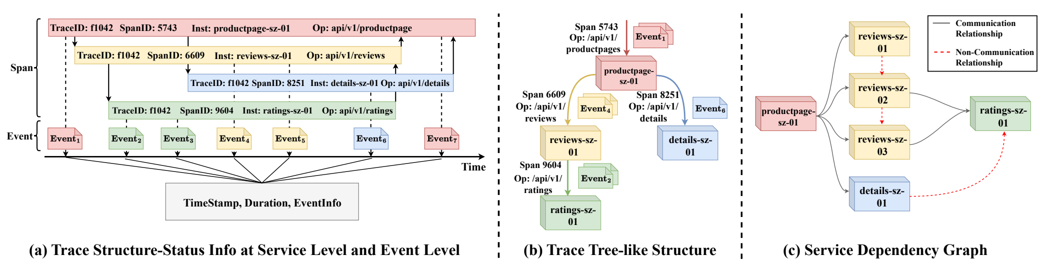


Fig. 2. An example of trace structure-status info and dependency graph info in a simple microservice system.

We provide a specific illustration of the two data sources(i.e., trace data and dependency graph) used in our method through an example microservice system. This example microservice system deploys Istio's Bookinfo application [13] and includes four pods and three nodes. *productpage-sz-01* is deployed on Node *A*, *reviews-sz-01*, *reviews-sz-02*, and *reviews-sz-03* are on Node *B*, while *details-sz-01* and *ratings-sz-01* are on Node *C*. Traces are generated and collected using the Opentelemetry framework [25]. Fig. 2(a) displays trace data generated from a single request. It consists of a series of spans, with events incorporated to enrich the information associated with each span. Microservice instances on a trace are interconnected via spans, forming a tree-like topology as depicted in Fig. 2(b). Aggregating multiple traces enables us to obtain communication-relationships within the system (as denoted by the black lines in Fig. 2(c)). Additionally, based on the system's deployment configuration files, we can identify non-communication relationships in the system (as indicated by the red lines in Fig. 2(c)).The rule for determining the direction of edges representing non-communication relationships is as follows: it points from instances with low resource utilization to those with high resource utilization.

2.2 Motivation

This section outlines the motivation behind our work, which aims to efficiently localize root causes at multi-levels by integrating in-depth trace-based analysis with overall dependency graph analysis.

2.2.1 ***Motivation 1: Enhancing RCA through trace-graph fusion.*** We investigate and summarize the extent of trace and graph analysis in current state-of-the-art RCA methods, as illustrated in Table 1. Most methods only consider either trace-structure-based or graph-based analysis. Despite

Table 1. Comparison of state-of-the-art RCA methods. The difference between "superficial" and "in-depth" trace analysis lies in whether fine-grained information (i.e., events) is considered. "commu" denotes communication relationship.

Approach	Trace Analysis		Graph Analysis		Analysis Level
	Superficial	In-depth	Commu	Non-commu	
MicroRCA [38]	✗	✗	✓	✓	Service
MicroScope [22]	✗	✗	✓	✓	Service
MicroRank [41]	✓	✗	✓	✗	Invocation
TraceAnomaly [23]	✓	✗	✗	✗	Invocation
TraceRCA [21]	✓	✗	✗	✗	Service
SBLD [30]	✗	✗	✗	✗	Log
PDiagnose [11]	✓	✗	✗	✗	Resource & Log
Eadro [16]	✓	✗	✓	✗	Service
TraGraphRCA	✓	✓	✓	✓	Service & Event

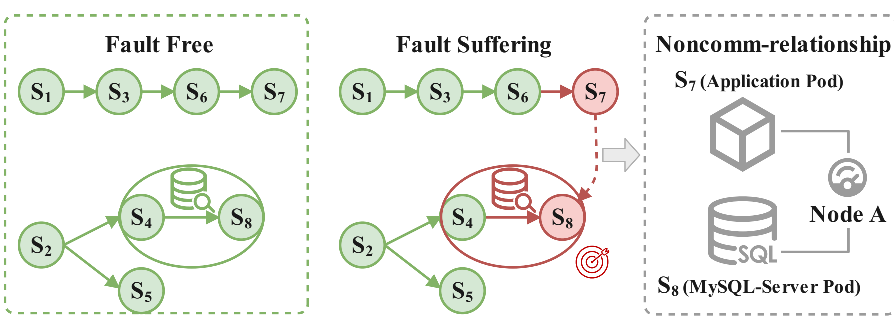


Fig. 3. An example of the importance of combining trace and graph from real-world case.

MicroRank [41] and Eadro [16] tends to cover both aspects. Unfortunately, their trace analysis is limited to just considering latency and graph analysis does not account for non-communication relationships. We implemented seven of these methods (excluding Eadro due to its supervised nature) and validated their RCA performance on two datasets. Our experimental results (§ 4.2.1) demonstrated that neither the existing trace-structure-based nor the graph-based approaches can provide satisfying RCA result.

We investigate a real-world issue from Huawei Cloud to further illustrate our point, as shown in Fig. 3. During the fault-suffering phase, the actual root cause of the issue was an overload attack on S_8 (a MySQL server pod), involving numerous requests containing full-table queries for S_8. This caused S_8 to consume a significant portion of CPU resources on Node A, dramatically slowing down S_7 (an application pod) that also runs on Node A. This slowdown resulted in a latency increase for traces passing through S_7.

For trace-structure-based approaches [21, 23], analyzing the differences between traces from fault-free and fault-suffering phases enabled them to pinpoint issues with S_7 and S_8. However, due to the lack of an overall analysis of the dependency graph, especially non-communication relationships, they treated S_7 and S_8 as separate root causes, failing to integrate them together. On the other hand, although graph-based approaches [22, 38] could identify relationships between

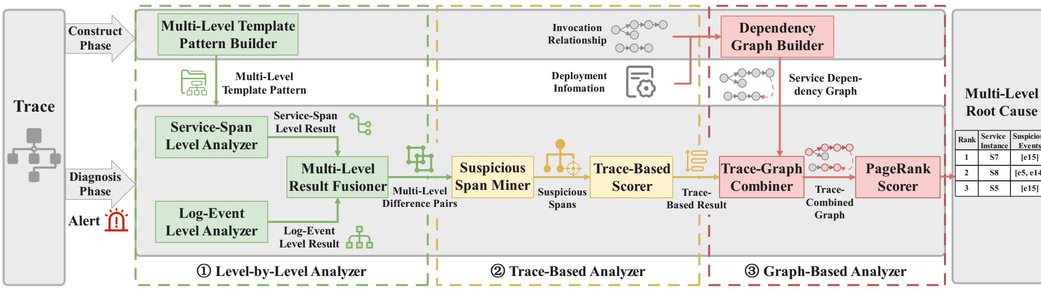


Fig. 4. System overview of *TraGraphRCA*.

S_7 and S_8, these methods which lack in-depth analysis focused on individual requests, could not detect the abnormal full-table queries from S_4 to S_8, missing the actual root cause.

The analysis above motivates us to design a more effective RCA method that combines in-depth trace-structure-based analysis and overall dependency-graph-based analysis, which aims not only to pinpoint root causes comprehensively but also to accurately coverage them.

2.2.2 ***Motivation 2: Localizing root cause at multi levels.*** Table 1 also shows the RCA level of the current approaches. Most of them identify root causes at the service level, lacking diagnostic details within the service. While SBLD [30] and PDiagnose [11] pinpoint the error logs, they are limited to specific types of logs. We emphasize the importance of providing fine-grained diagnostic results in the RCA process. For instance, consider S_8 in Fig. 3, which handles an average of 50,000 requests per hour, resulting in more than 3,000,000 events. Even after identifying the root cause service (S_8), SREs still face a significant challenge in pinpointing the specific root cause events (i.e., the full table query SQL events). Hence, a multi-level root cause analysis involving services and events is essential to automate this process.

2.3 Problem Formulation

We formalize the problem of multi-level RCA in a microservice system by combining trace and graph information as follows. Given a time window $\mathcal{W}$ (default 5 minutes) at normal system states, we collect the sets of traces $T = \{T_1, ..., T_t\}$ and log events $E = \{E_1, ..., E_e\}$. Each log event E_i is associated with a trace ID and a corresponding trace T_j. We first aggregate the normal trace at service-span level and log-event level, constructing cross-level normal template patterns $\mathcal{T}$. Additionally, we build a service dependency graph $\mathcal{G}$ based on the service communication and non-communication relationships, storing in the form of bitmaps.

When an alarm occurs in the microservice system, we obtain a set of suspicious traces $T' = \{T'_1, ..., T'_{t'}\}$ and events $E' = \{E'_1, ..., E'_e\}$ within a time window (e.g., 5 minutes). The primary object of multi-level RCA is to determine the root cause service instances and log events of the alarm. To achieve this object, we formalize multi-level RCA based on a parameterized model $\mathcal{F} : (\mathcal{T}, \mathcal{G}, T', E') \rightarrow (\mathcal{S}, \mathcal{L})$, where $\mathcal{S}$ represents the root cause service instances and $\mathcal{L}$ represents the root cause log events.

3 Methodology

Motivated by the above motivations, we propose a practical multi-level RCA approaches, namely *TraGraphRCA*, which combine both in-depth trace-structure-based analysis and dependency-graph-based analysis to localize root causes. Fig. 4 shows the overall architecture of *TraGraphRCA*, containing two phases: construct phase and diagnose phase. In construct phase, *TraGraphRCA* constructs multi-level normal template patterns and dependency graph from normal traces and configuration files. In diagnose phase, *TraGraphRCA* has three main parts: ①*Level-by-level Analyzer*, ②*Trace-Based Analyzer* and ③*Graph-Based Analyzer*.

3.1 The Construction Phase

In construction phase, we first obtain all traces and events within a time window $\mathcal{W}$ in the fault-free phase of the system. Obtaining normal data from a fault-free time window is easy because most of the time of production environment is in a fault-free phase [19].

Construct normal template patterns. During the fault-free phase, we cluster and extract trace template patterns at both the service and event levels to model the system's normal behavior. (1) At service-span level, traces with the same service-span tree structure (described in 2.1) are grouped into a cluster. This tree structure serves as the basis for the corresponding extracted template pattern for that cluster. Additionally, we calculate the upper bound for the normal duration of each span based on the traces used to build these patterns. We utilize the 3-σ principle [17], commonly used for outlier detection [22, 41], to compute this normal upper bound, represented as $ub_s = \mu_0 + 3 * \sigma_0$, where μ_0 denotes to the average duration, and σ_0 represents the standard deviation. This upper bound is then attached to the template pattern. The green part of Fig. 5 illustrates the process of constructing service-span level template patterns. (2) At log-event level, we aggregate event sequences occurring on the same span in traces. The resulting template pattern records the possible events at each position in the event sequence. Similar to the service-span level pattern, we also calculate the duration upper bound for each event using the 3-σ principle, denoted as ub_e. The green part of Fig. 6 illustrates the process of constructing log-event level template patterns.

Construct service dependency graph. During the construction phase, we gather communication relationships between services by aggregating trace data, and obtain non-communication relationships from Configuration Management Database (CMDB). Based on these relationships, we build a service dependency graph, as described in 2.1. To improve the efficiency, we store and maintain the graph in the form of bitmaps [14].

3.2 Level-by-Level Analyzer

During the fault-suffering phase, traces would exhibit structural or state information differences compared to traces from the fault-free phase. For example, in Fig. 5, p^s_{t1} shows a broken link, and the span on p^s_{t3} experiences increased delay. These differences serve as crucial evidence for RCA. To capture them, Level-by-Level Analyzer conducts analysis on suspicious traces at both the service-span level and log-event level. At each level, Level-by-Level Analyzer extracts information from each suspicious trace to generate suspicious pattern $\boldsymbol{p}_t$, and matches it with the corresponding normal template pattern $\boldsymbol{p}_n$. It then detects the differences between $\boldsymbol{p}_t$ and $\boldsymbol{p}_n$, and outputs the difference pairs $\boldsymbol{D}$ for further analysis.

3.2.1 *Service-Span Level Analyzer*.

At this level, we focus on service instances involved in traces and their invocations (spans).

Get matched template pattern at service-span level. In the diagnosis phase, we extract the service-span level pattern p^s_t from each suspicious trace in a manner similar to the construction phase. We then find the most similar normal template pattern that matches p^s_t. Due to anomalies in the microservice system, the suspicious trace may undergo structural changes, such as broken links, we cannot always find a normal template pattern that exactly matches the suspicious trace pattern. To assess the degree of similarity between two service-span level patterns, we define the service-span Jaccard similarity.

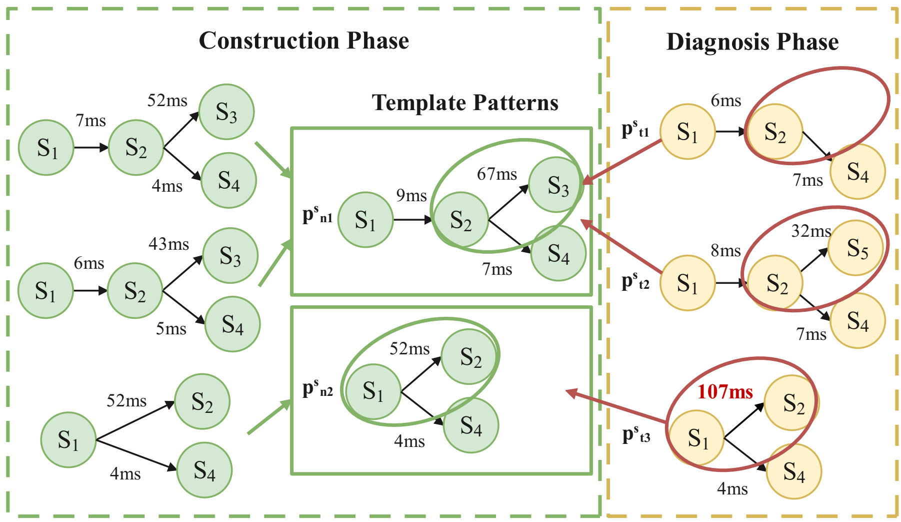


Fig. 5. Comparison between the service-span structure of the suspicious trace and the constructed normal template.

Definition 1 (Service-Span Jaccard Similarity J_s). *For suspicious pattern p_t^s and normal template pattern p_n^s, we denote the set of spans for p_t^s as SP_t and the set of spans for p_n^s as SP_n. Their service-span Jaccard similarity is calculated as $J_S(p_t^s, p_n^s) = \frac{|SP_t \cap SP_n|}{|SP_t \cup SP_n|}$.*

For example, in Fig. 5, the intersection of p_{t1}^s and p_{n1}^s spans is $\{(S_1 \longrightarrow S_2), (S_2 \longrightarrow S_4)\}$, the union of spans is $\{(S_1 \longrightarrow S_2), (S_2 \longrightarrow S_3), (S_2 \longrightarrow S_4)\}$. Therefore, their service-span Jaccard similarity J_S is $\frac{2}{3}$. Similarly, the J_S between p_{t1}^s and p_{n2}^s is $\frac{1}{3}$. Therefore, p_{n1}^s is the most closely matched normal template pattern for p_{t1}^s.

Mine difference pairs at service-span level. Once the most closely matched template pattern p_m^s of suspicious pattern p_t^s is found, we examine the structural differences and duration differences between p_t^s and p_m^s. (1) Structural differences refer to spans at the same position where p_t^s and p_m^s differ (such as p_{t1}^s and p_{n1}^s in Fig. 5). (2) Duration differences refer to a matched span pair (s_i^t, s_i^m) between p_t^s and p_m^s, where the duration d_i^t of s_i^t exceeds the normal upper bound ub_{si} recorded by s_i^m (such as p_{t3}^s and p_{n2}^s in Fig. 5). When differences are detected, we record a set of service-span level difference pairs $D_s = \{(s_1^t, s_1^m), ..., (s_n^t, s_n^m)\}$. Regarding structural differences, it is possible for one item in the difference pair to be empty. In such cases, we fill it with an empty value denoted as $\boldsymbol{n}$.

3.2.2 *Log-Event Level Analyzer*.

As mentioned in § 2.2.2, reporting root causes at the service-span level is insufficient for SREs. To conduct a more detailed analysis, we examine the structure and status information of log events (as described in § 2.1) on each span.

Mine difference pairs at log-event level. To analyze information at log-event level, we compare the event sequence $\{e_1^t, ..., e_n^t\}$ (e.g., $\{e_8, e_3, e_9\}$ in Fig. 6) on suspicious trace p_t^s with the corresponding normal event template pattern (e.g., $P_{S_2 \rightarrow S_3}$). As shown in Fig. 6, we analyze the log-event differences in structure and duration as follows: (1) Structural difference refers to the event e_i^t that is not a part of the possible event set in template pattern (such as e_8 and e_9 of p_{t1}^s not in $P_{S_2 \rightarrow S_3}$). (2) Duration difference arises when the duration of e_i^t exceeds the upper bound recorded

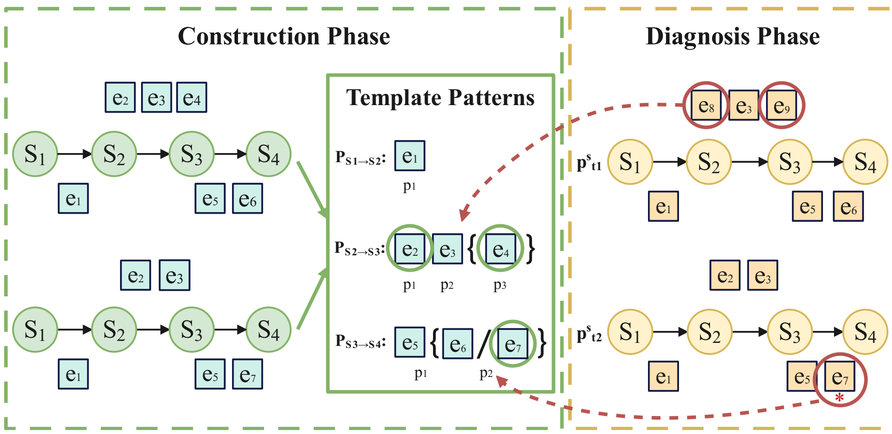


Fig. 6. Comparison between the log-event information of suspicious traces and the normal templates constructed during the construction phase ("*" indicates a latency increase).

Table 2. An example of trace-based analysis for Fig. 7.

Suspicious Span	Support		$I_{\mathcal{SS}}$	$Score_{\mathcal{SS}}$	Suspicious Events
	$\mathcal{S}p_{\mathcal{E}}$	$\mathcal{S}p_{\mathcal{V}}$			
$S_1 \longrightarrow S_3$	2	1	0	0.33	$[e_1]$
$S_3 \longrightarrow S_6$	1	2	1	1.33	$[e_{12}, e_{13}]$
$S_5 \longrightarrow S_7$	0	1	0	1	$[e_{15}]$
$S_6 \longrightarrow S_8$	0	1	2	3	$[e_5, e_{14}]$

on the template (such as e_7 of p^s_{t2}). Once all differences are detected, we record a set of log-event level difference pairs $D_e = \{(e^t_1, e^m_1), ..., (e^t_n, e^m_n)\}$.

3.2.3 ***Multi-level Result Fusion.*** After obtaining the difference pairs D_s and D_e at two levels, we merge D_e into D_s as follows: For each $d_{si}(s^t_i, s^m_i) \in D_s$, we combine all $d_{ej}(e^t_j, e^m_j) \in D_e$ that satisfy e^t_j within the span s^t_i to form a set $\boldsymbol{S}_{ei}$, which is then merged into $d_{si}(s^t_i, s^m_i)$. This process results in a multi-level difference pair $d(s^e, s^v, \boldsymbol{S}_e)$, representing the transition of the fault-free phase template pattern s^e to the fault-suffering phase pattern s^v, in other words, s^e denotes the expected span pattern and s^v denotes the violated. The set $\boldsymbol{S}_e$ records the differences at log-event level within this span. We use D to denote the set of all multi-level difference pairs mined by Level-by-Level Analyzer.

3.3 Trace-Based Analyzer

After obtaining the difference pair set $\boldsymbol{D}$ generated during the diagnosis phase, SREs need to determine which differences are more likely to reflect the root cause. To address this, we designed *Trace-Based Analyzer*, which leverages $\boldsymbol{D}$ to uncover a set of suspicious span $\mathcal{SS}$, and employs trace-based analysis to score the likelihood of each suspicious span $\boldsymbol{ss}$ reflecting the actual root cause.

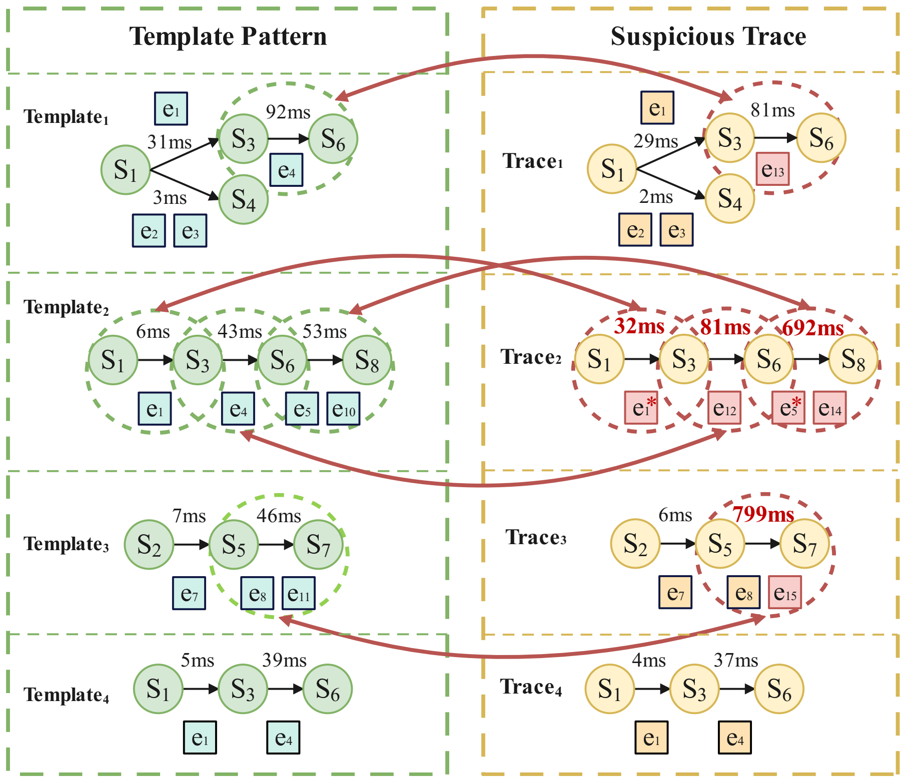


Fig. 7. Examples of difference pairs between template patterns and suspicious traces. "*" indicates a latency increase.

3.3.1 ***Suspicious Span Miner***.

For difference pair $d_i(s_i^e, s_i^v, S_{ei})$, we can extract the expected span s_i^e and the violated span s_i^v. As shown in Fig.7, the spans enclosed by the green dashed circle are the expected spans, while the spans enclosed by the red dashed circle are the violated spans. We consider the set of all expected spans s_i^e as the collection of suspicious spans, denoted as $\mathcal{SS} = \{\boldsymbol{ss}_1, ..., \boldsymbol{ss}_n\}$. These spans, which undergo changes during the diagnosis phase, likely reflect the impact of anomalies on the system. Therefore, in Fig. 7, the suspicious span set is $\mathcal{SS} = \{S_1 \rightarrow S_3, S_3 \rightarrow S_6, S_5 \rightarrow S_7, S_6 \rightarrow S_8\}$. For each suspicious span $\boldsymbol{ss}$, we identify its violated events according to the event level information in the difference pair. The violated events are considered as suspicious events on $\boldsymbol{ss}$. For example, in terms of $S_3 \rightarrow S_6$ in Fig. 7, both e_{12} and e_{13} are considered as its suspicious events.

Definition 2 (Influence I). *We use U_t to denote the upstream span set of suspicious span $\boldsymbol{ss}$ on trace t, and denote the violated span set on trace t as V_t. The influence of $\boldsymbol{ss}$ is denoted as the maximum of the number of violated spans in U_t for each trace t_i, i.e., $I(\boldsymbol{ss}) = max(|U_{t_i} \cap V_{ti}|)$. We use $\mathcal{I_{SS}}$ to denote the influence set of all suspicious spans at diagnosis phase.*

Definition 3 (Support sp). *Given the count set $C_s = \{c_1, ..., c_k\}$ of a span pattern s, where c_i denotes s occurs c_i times on trace t_i. The support $sp(s)$ of span pattern s is the sum of the counts in all traces, i.e., $sp(s) = \sum_{i=0}^{k} c_i$. For a suspicious span $\boldsymbol{ss}$, let $\boldsymbol{e}$ to be its expected pattern and let $\boldsymbol{v}$ to be its violated pattern. $sp(\boldsymbol{e})$ and $sp(\boldsymbol{v})$ denote the support of $\boldsymbol{e}$ and $\boldsymbol{v}$ at diagnosis phase, respectively. We* *use $\mathcal{S}p_{\mathcal{E}}$ and $\mathcal{S}p_{\mathcal{V}}$ to denote the support set of expected and violated pattern of all suspicious spans at diagnosis phase, respectively.*

After extracting suspicious spans, *Suspicious Span Miner* counts the occurrences of the expected pattern and violated pattern of each suspicious span to calculate its support. It also counts the number of violated spans upstream of each suspicious span to calculate its influence. Table 2 shows an example of calculating influences and supports in Fig. 7. For suspicious span $S_3 \longrightarrow S_6$, its expected pattern occurs in one trace ($Trace_4$) and its violated pattern occurs in two traces ($Trace_1$ and $Trace_2$) in diagnosis phase. Therefore, $\mathcal{S}p_{\mathcal{E}}(S_3 \longrightarrow S_6) = 1$, $\mathcal{S}p_{\mathcal{V}}(S_3 \longrightarrow S_6) = 2$. For calculation of influence, suspicious span $S_3 \longrightarrow S_6$ does not have any violated spans upstream in $Trace_1$, but has one in $Trace_2$. Thus we calculate $I_{\mathcal{D}}(S_3 \longrightarrow S_6)$ as the maximum of 0 and 1, which is equal to 1

3.3.2 ***Trace-Based Scorer***. *Trace-Based Scorer* aims to assess the contribution of each suspicious span to root cause diagnosis. It is built on two core ideas: (1) In the diagnosis phase, suspicious spans that appear multiple times as violated pattern but rarely as expected pattern are more likely to reflect the root cause. (2) Within an abnormal propagation chain, suspicious spans that have more violated downstream (cause more spans to be violated) are more likely to reflect the root cause. For each suspicious span ss, *Trace-Based Scorer* compute its ranking score $Score_{\mathcal{SS}}$ as follows.

$$Score_{\mathcal{SS}}(ss) = \frac{\mathcal{S}p_{\mathcal{V}}(ss)}{\mathcal{S}p_{\mathcal{E}}(ss) + \mathcal{S}p_{\mathcal{V}}(ss)} * (1 + I_{\mathcal{SS}}(ss)). \tag{1}$$

The $Score_{\mathcal{SS}}(ss)$ of a suspicious span ss combines the two core ideas mentioned above to evaluate its contribution to root cause diagnosis. We use multiplication (" $\times$ ") rather than addition(" $+$ ") to combine the two results since they are on different scales. As an example, in Fig. 7, a code exception occurs on S_8. From the figure, we can observe that the suspicious span $S_6 \longrightarrow S_8$ only exhibits violation in the diagnosis phase without occurrence as an expected pattern. Additionally, on $trace_2$, both upstream spans of $S_6 \longrightarrow S_8$ are violated. Intuitively, we can infer that $S_6 \longrightarrow S_8$ is more likely to reflect the root cause. In terms of *Trace-Based Scorer*, $Score_{\mathcal{S}}(S_6 \longrightarrow S_8) = \frac{1}{0+1} * (1+2) = 3$, which is the highest score in the example.

3.4 Graph-Based Analyzer

The previous trace-based analysis provided the suspicious span set $\mathcal{SS}$ with suspicious events and the trace-based $Score_{\mathcal{SS}}$. However, as mentioned in section 2.2.1, trace-based analysis lacks the exploration of overall dependencies between services, especially non-communication dependencies. To solve this, we designed *Graph-Based Analyzer* to further analyzes the suspicious spans through combining with the service dependency graph. *Graph-Based Analyzer* constructs a trace-combined dependency graph and utilizes a custom PageRank [27] on it. It outputs a list of ranked root cause service instances associated with the suspicious log events as multi-level root cause analysis results.

3.4.1 ***Trace-Graph Combiner.*** To combine trace and graph analysis, *Graph-Based Analyzer* integrates the score of the suspicious span set $Score_{\mathcal{SS}}$ as edge weights into the service dependency graph, resulting in a trace-combined service dependency graph.

Definition 4 (Trace-combined Service Dependency Graph $G_{\mathcal{T}}$). *The trace-combined service dependency graph $G_{\mathcal{T}} = \langle V, E \rangle$ is a directed graph consisting of n nodes (service instances) and m edges (dependencies between instances). If there is a dependency from node s to t, the edge $\langle s, t \rangle$ will be included in E. If $s \longrightarrow t$ is a suspicious span, the edge weight $w_{\langle s,t \rangle} = 1 + Score_{\mathcal{SS}(s \rightarrow t)}$; otherwise, $w_{\langle s,t \rangle} = 1$.*

For example, in Fig. 8, the edge weight of $\langle S_1, S_3 \rangle$ is $w_{\langle S_1, S_3 \rangle} = 1 + Score_{\mathcal{SS}(S_1 \rightarrow S_3)} = 1.33$. On the other hand, $S_1 \longrightarrow S_4$ and $S_8 \longrightarrow S_7$ are not suspicious spans, so their edge weight is 1.

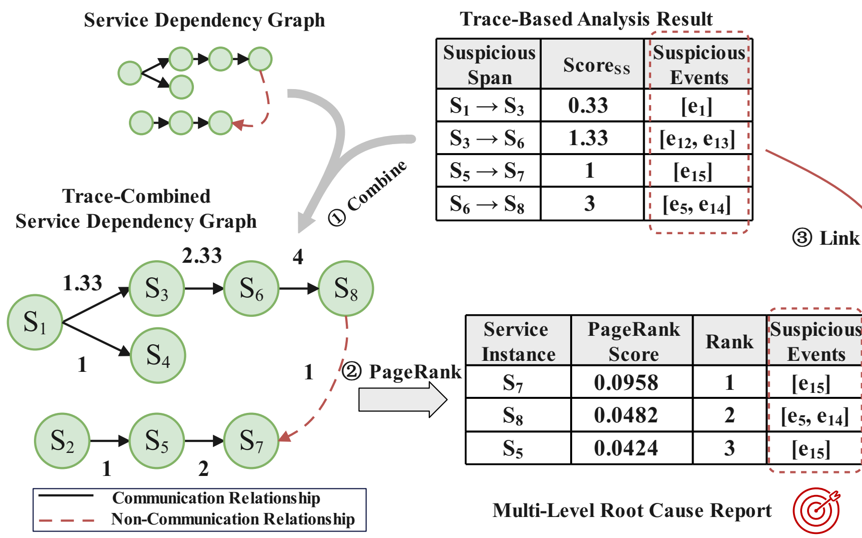


Fig. 8. An example of combining trace-based result and graph analysis to get a multi-level root cause report.

3.4.2 ***PageRank Scorer.*** After obtaining the trace-combined service dependency graph $G_{\mathcal{T}}$, we apply a customized PageRank [27] on $G_{\mathcal{T}}$ to calculate the PageRank scores for each node. $\mathbf{A}_{st}$ is defined as the probability that a walk starting from s terminates at t, which reflects the importance of t with respect to s. After considering the edge weights, the value of $\mathbf{A}_{st}$ can be calculated by:

$$\mathbf{A}_{st} = \begin{cases} \frac{w_{\langle s,t \rangle}}{\sum w_{\langle s,O(s) \rangle}}, & t \in O(s) \\ 0, & \text{otherwise} \end{cases}, \tag{2}$$

where $O(s)$ denotes the out-neighbors of s. All of the $\mathbf{A}_{st}$ will be combined into the transition matrix $\mathbf{A}$. To avoid getting trapped in local traps [2], an escape matrix $e = [\frac{1}{n}, ..., \frac{1}{n}]$ is incorporated to allow for a probability of randomly jumping out, following previous approach [2]. Therefore, the equation of the qth iteration in PageRank iterative process [27] is defined as:

$$\mathbf{v}^{(q)} = d\mathbf{A}\mathbf{v}^{(q-1)} + (1-d)\mathbf{e}. \tag{3}$$

where d is the damping factor ($0 \leq d \leq 1$, default $d = 0.85$ in this paper), the solution $\mathbf{v}$ is initialized as $\left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$. After each iteration, we are gradually approaching a more accurate estimation of the final value. The outcome vector represents the scores assigned to each node in a ranked order, e.g., the table on the right side of Fig. 8 shows the outcome after applying PageRank to the trace-combined service dependency graph on the left.

Generate multi-level root cause analysis results. The ranking of service instances obtained by PageRanker represents the likelihood of each instance being the root cause. Trace and graph analysis are leveraged comprehensively. For example, in Fig. 8, S_7 executes a slow SQL query, causing a sudden increase in CPU usage on the node. S_8 runs on the same node as S_7, resulting in the failure propagating to S_8. *Graph-Based Analyzer* considers the non-communication dependency from S_8 to S_7 and correctly identifies S_7 as the top-ranked root cause node. If only trace-based analysis is used, the root cause would easily be misattributed to S_8, because it is at the end of the trace and triggers multiple exceptions, resulting in the highest trace-based score. Ultimately, for each root cause instance, *TraGraphRCA* attaches suspicious events from spans where this instance is a caller or callee to the root cause analysis report, yielding a multi-level root cause report from which SREs can obtain root cause instances and specific root cause events.

4 Experimental Evaluation

To evaluate the effectiveness and efficiency of *TraGraphRCA*, we aim at answering the following research questions (RQs).

- **RQ1:** How accurate is the multi-level root cause analysis of *TraGraphRCA*?
- **RQ2:** How much does the fusion of trace-structure-based analysis and dependency-graph-based analysis contribute to the effectiveness of *TraGraphRCA*?
- **RQ3:** How efficient is the analysis of *TraGraphRCA*? To what extent does the use of bitmaps improve its efficiency?

4.1 Experimental Setup

4.1.1 ***Dataset $\mathcal{A}$ Setup.*** Dataset $\mathcal{A}$ consists of 54 reliability or performance issues, involving 13 large-scale microservice systems (each system contains 284 services on average) and 1327 physical nodes from Huawei Cloud. These issues occurred between April 2023 and June 2023. The types of failures include CPU exhausted, memory exhausted, network delay, slow SQL execution, code exception, and failed third-party package calls. The labeled root causes were identified by SRE and domain-specific technical experts, using two levels of granularity: service-span and log-event. The first level identifies the root cause node where the failure occurred, while the second level captures the events reflecting the actual root cause during the failure period. The dataset was collected through an Application Performance Management (APM) platform, including trace data and relevant metrics(e.g., Average Response Time, QPS, Error Rate).

4.1.2 ***Dataset $\mathcal{B}$ Setup.*** **Microservice Benchmark.** Dataset $\mathcal{B}$ is based on two widely-used open-source microservice systems, namely OnlineBoutique [8] and TrainTicket [6], which have been extensively studied in previous research [5, 18, 41, 44, 46, 49]. OnlineBoutique is a web-based e-commerce application consisting of 10 microservices implemented in various programming languages and communicating with each other using gRPC. TrainTicket offers a railway ticketing service that involves 45 services communicating through synchronous REST invocations and asynchronous messaging.

Experimental Platform. We have deployed the OnlineBoutique and TrainTicket applications on a Kubernetes [1] platform consisting of 12 virtual machines. Each virtual machine is equipped with an 8-core 2.10GHz CPU, 16GB of memory, and runs on the Ubuntu 18.04 operating system. To collect traces, we utilize Opentelemetry Collector [25], which stores them in Grafana Tempo [9].

Data Collection. To simulate latency and reliability issues in a microservice system, we utilized Chaosblade [4] to inject a total of 56 faults into these two benchmark microservices. These faults encompassed various types including CPU exhausted, memory exhausted, network delay, code exceptions, and error returns. The ground truths refer to the known injected pods or code regions and the types of faults injected. A summary of faults in our datasets is shown in Table 3. An overview of our experimental datasets can be found in Table 4.

4.1.3 ***Baselines.*** We employ seven state-of-the-art methods as baselines, which consist of three dependency-graph based approaches (i.e., MicroRCA [38], MicroScope [22], MicroRank [41]), two trace-structure based approaches (i.e., TraceRCA [21], TraceAnomaly [23]), one log-event based approach (i.e., SBLD [30]), and one method that incorporates multi-modal data (i.e., PDiagnose [11]). To evaluate the contribution of combining trace and graph, we create the two variants of *TraGraphRCA* (i.e., *TraGraphRCA* w/o $\mathcal{T}$, *TraGraphRCA* w/o $\mathcal{G}$) and conduct ablation experiments [31].

Table 3. Summary of faults in the datasets.

Fault Type	Description	Case Number
CPU Exhausted	CPU Exhausted refers to a system failure caused by the depletion of available CPU (Central Processing Unit) resources. This issue commonlyarises due to misconfigurations or excessive processing demands.	17
Memory Exhausted	Memory Exhausted refers to a system failure caused by the depletion of available memory resources. This situation commonly arises due to misconfigurations, excessive data load, or software inefficiencies.	11
Network Delay	Network Delay occurs when there is a slowdown in the transmission of network packets, which causes long latency. This can happen due to network congestion or high traffic volume.	36
Error Return	Error Return refers to a situation where an application encounters errors and return wrong responses. This can occur due to factors such as software bugs, invalid input, or incorrect configurations	8
Code Exception	Code Exception refers to a scenario where a software program encounters unexpected conditions or situations during its execution, leading to an interruption in the normal flow of the program.	8
Failed Third-party Package Calls	Failed Third-party Package Calls occur when an application attempts to utilize external libraries or packages but encounters errors or failures during the execution of these calls.	6
Slow SQL Execution	Slow SQL Execution refers to a situation where database queries take an unusually long time to process. This issue can arise due to an overload attack, poorly optimized queries, or high server loads.	14

Table 4. Experiment datasets overview.

Dataset	Microservice Benchmark	Trace Number	Fault Number	Fault Type Number
Dataset $\mathcal{A}$	13 Production Microservice systems	792,403	54	6
Dataset $\mathcal{B}$	OnlineBoutique and TrainTicket	114,036	56	5

For the baseline implementations, MicroRank [41], TraceRCA [21], TraceAnomaly [23], and PDiagnose [11] offer open-source versions that we directly utilize. MicroRCA [38], MicroScope [22], and SBLD [30] lack open-source implementations, prompting us to create our own versions. To ensure accuracy, we closely adhere to the methods described in related papers and employ the exact libraries they used. For SBLD and PDiagnose, methods requiring log data sources, we treat log events on traces as the input log data. Considering that they are designed to pinpoint error logs rather than trace events, their results would be determined to be correct if their output error logs lie in the root cause events. The details of the seven baselines are as follows.

- MicroRCA [38] is a dependency-graph based approach that utilizes the PageRank algorithm to identify suspicious services by analyzing extracted abnormal subgraphs.
- MicroScope [22] is a dependency-graph based approach that identifies root causes by analyzing the correlation of metrics within a dependency framework.
- MicroRank [41] is a mainly dependency-graph based approach that combines the personalized Pagerank method with the Spectrum method to locate suspicious root causes. It utilizes traces only to examine latency and does not conduct analysis on individual requests.

- TraceRCA [21] is a trace-structure based approach that identifies root cause services by analyzing the proportion of normal and abnormal traces on services and calculating the in-set score.
- TraceAnomaly [23] is a trace-structure based approach that utilizes deep learning to offline learn normal trace patterns and online detect abnormal traces for root cause analysis.
- SBLD [30] is a log-event based approach that utilizes Spectrum algorithms to identify root cause log events.
- PDiagnose [11] takes metrics, traces, and logs as inputs, converts them into time series, and identifies the root cause by voting for the abnormal time series.

The two variants of *TraGraphRCA* are implemented as follows.

- *TraGraphRCA* w/o $\mathcal{T}$ is the variant that only performs dependency-graph based analysis with a modification to simulate random walks by using an equal probability transition instead of the trace-score-based node transition strategy.
- *TraGraphRCA* w/o $\mathcal{G}$ is the variant that solely relies on trace-structured based analysis result without traversing the dependency graph.

4.1.4 ***Evaluation Metrics.*** To assess the effectiveness of *TraGraphRCA*'s multi-level analysis, we utilized the following four metrics, where I is the set of latency or reliability issues.

- **Top-k accuracy (**$A@k$**)** represents the probability that the true root cause is included in the top-k positions of the results. Let rc_i be the root cause of the i-th issue, $Rank_i^k$ be the top-k result list for the ith issue. $A@k$ is calculated as: $A@k = \frac{1}{|I|}\sum_{i=1}^{|I|}\left(rc_i \in \mathrm{Rank}_i^k\right)$. We use $AS@k$ and $AE@k$ to represent the top-k accuracy at the service level and the event level, respectively.
- **Mean reciprocal rank (**MRR**)** represents the inverse of the rank of the first identified root cause. If the actual root cause is not present in the result list, its reciprocal rank is considered to be zero. Let r_i be the rank of the root cause in the returned list for the ith issue. The calculation for MRR is: $MRR = \frac{1}{|I|}\sum_{i=1}^{|I|}\frac{1}{\mathrm{rs}_i}$. We use $MRRS$ and $MRRE$ to represent the mean reciprocal rank at the service level and the event level, respectively.

4.2 Evaluation Results

4.2.1 ***RQ1: Effectiveness of TraGraphRCA at multiple levels.*** **Effectiveness at service-span level.** Table 5 shows the effectiveness evaluation results of different approaches for root cause analysis at service-span level. From Table 5, it is evident that *TraGraphRCA* outperforms other baseline methods across all three metrics on both datasets. The remarkable accuracy of *TraGraphRCA* in identifying root causes at the service-span level can be attributed to its integration of trace-structure-based analysis and dependency-graph-based analysis. Unlike other methods that mostly only conduct single-dimensional analysis, *TraGraphRCA* takes into account more factors, resulting in superior analysis effectiveness.

After a detailed analysis of the root cause analysis results from various methods, we have reached further conclusions. For trace-structure-based methods like TraceRCA and TraceAnomaly (with an average $MRRS$ of 0.51) conduct individual analysis of trace structures but lack the ability to uncover non-communication relationships and thus unable to identify resource consumption anomalies. For dependency-graph-based methods like MicroScope, MicroRCA, and MicroRank (with an average $MRRS$ of 0.43), they excel at analyzing anomalies that propagate across service dependencies. However, these methods fail to detect exceptions caused by code exceptions that break the trace chain, leading to inaccurate root cause identification. SBLD (with an average $MRRS$ of 0.52) analyzes events from a spectrum perspective but also falls short in recognizing resource consumption anomalies. PDiagnose (with an average $MRRS$ of 0.58) integrates multi-modal data

Table 5. Comparison of baselines at service-span level.

Approach	DataSet $\mathcal{A}$			DataSet $\mathcal{B}$		
	AS@1	*AS*@3	*MRRS*	*AS*@1	*AS*@3	*MRRS*
MicroRCA [38]	29.63	57.41	0.4607	17.86	32.14	0.2054
MicroScope [22]	50.00	79.63	0.6578	25.00	39.29	0.2976
MicroRank [41]	42.59	77.78	0.6056	25.00	53.57	0.3690
TraceRCA [21]	48.15	72.22	0.6328	28.57	42.86	0.3512
TraceAnomaly [23]	51.85	81.48	0.6820	32.14	46.43	0.3750
SBLD [30]	55.56	75.93	0.6615	28.57	53.57	0.3869
PDiagnose [11]	50.00	77.78	0.6550	39.29	64.29	0.5000
TraGraphRCA w/o $\mathcal{T}$	61.11	88.89	0.7472	35.71	53.57	0.4286
TraGraphRCA w/o $\mathcal{G}$	62.96	81.48	0.7519	67.86	71.43	0.6905
TraGraphRCA	**85.19**	**94.44**	**0.8967**	**82.14**	**92.86**	**0.8851**

for analysis. However, its performance in Dataset $\mathcal{B}$ with an *AS*@1 below 40% could be attributed to its simplistic voting mechanism used for ranking.

Effectiveness at log-event level. At log-event level, we chose SBLD and PDiagnose as baselines because only these two approaches can perform root cause analysis at the log-event level. Table 6 shows the effectiveness of different approaches in root cause analysis at log-event level on both datasets. It can be observed that *TraGraphRCA* outperforms baselines across all the three metrics. Specifically, *TraGraphRCA* achieves an *AE*@1 of over 75%. This superior performance is attributed to its multi-level analysis of traces and the incorporation of dependency graphs for RCA.

On the other hand, SBLD locates root causes based on the frequency of abnormal events. However, in real-world scenarios, non-root cause nodes can also generate a significant number of faulty events due to fault propagation, leading to lower accuracy in SBLD's root cause localization. PDiagnose, in Dataset $\mathcal{A}$, exhibits low *AE*@1 and *AE*@3 values, both below 10%. This could be attributed to the fact that Dataset $\mathcal{A}$ consists of fault data from a large-scale production microservice system with complex service dependencies. PDiagnose does not analyze service dependencies and relies solely on a voting mechanism for root cause prioritization, resulting in lower accuracy in root cause localization.

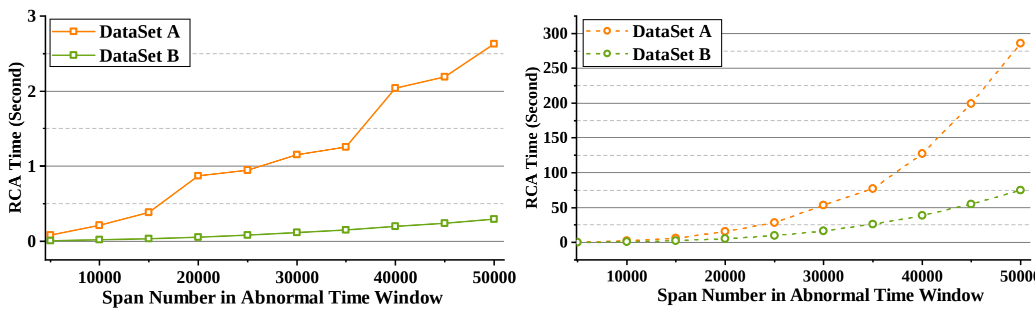
Conclusion. It is clear that *TraGraphRCA* outperforms the baselines across various metrics at both the two level. At the service-span level, *TraGraphRCA* an average improvement of 39.01% to 59.92% in *AS*@1. At log-event level, *TraGraphRCA* demonstrates an average improvement of 61.54% to 76.92% in *AE*@1 compared to the baseline method. The practical application results demonstrated in Appendix 7 also illustrate its excellent effectiveness.

4.2.2 ***RQ2: Contribution of the Combination of Trace and Graph.*** The last three rows of Table 5 and Table 6 demonstrate the results of the ablation experiments [31]. It is evident that *TraGraphRCA* achieves the best results across all metrics, indicating that both dependency-graph based analysis and trace-structured based analysis contribute to better root cause analysis.

After a thorough analysis of the results from both variants of root cause analysis, we have drawn further conclusions. On average, *TraGraphRCA* w/o $\mathcal{G}$ demonstrates higher accuracy in pinpointing the root cause compared to *TraGraphRCA* w/o $\mathcal{T}$. This is due to the utilization of multi-level analysis in trace-based analysis, allowing for more fine-grained diagnostics of common root causes such as network delay and code exceptions. However, when it comes to cross-trace propagated

Table 6. Comparison of baselines at log-event level.

Approach	DataSet $\mathcal{A}$			DataSet $\mathcal{B}$		
	AE@1	*AE@3*	*MRRE*	*AE@1*	*AE@3*	*MRRE*
SBLD [30]	19.23	57.69	0.4212	16.07	35.71	0.1976
PDiagnose [11]	3.85	7.69	0.0608	17.86	37.50	0.2243
TraGraphRCA w/o $\mathcal{T}$	57.69	69.23	0.6577	19.64	39.29	0.2742
TraGraphRCA w/o $\mathcal{G}$	61.54	65.38	0.6454	66.07	67.86	0.5963
TraGraphRCA	**80.77**	**92.31**	**0.8623**	**78.57**	**89.29**	**0.8364**



(a) TraGraphRCA's Diagnose Efficiency (b) TraGraphRCA w/o Bitmaps's Diagnose Efficiency

Fig. 9. The efficiency with and without use of bitmaps.

anomalies like resource shortages, *TraGraphRCA* w/o $\mathcal{T}$ performs better. This is because the service dependency graph captures the dependencies between services, including non-communication relationships. This finding further validates the motivation, as described in § 2.2.1, of combining trace and graph to enhance the effectiveness of root cause analysis.

4.2.3 ***RQ3: Efficiency of TraGraphRCA and Contribution of bitmaps.*** Efficiency is a crucial factor in determining the applicability of root cause analysis algorithms in real-world production. The efficiency of *TraGraphRCA* in root cause analysis heavily relies on the number of spans within the diagnosis phase. To analyze the efficiency and scalability of *TraGraphRCA* with different span sizes, we conducted an experiment to observe the changes in diagnosis time as the span number increased. Additionally, we performed an ablation experiment [31] to validate the contribution of using bitmaps [14] to maintain the dependency graph. It is important to note that the time taken to construct template patterns and dependency graph was not included in the diagnosis time calculation, because the construction of these two components is incrementally performed during the normal operation of the system and they can be reused multiple times during the diagnosis process.

Fig. 9 shows the significant improvement in the root cause analysis efficiency of *TraGraphRCA* compared to *TraGraphRCA* w/o bitmaps. The use of bitmaps reduces the diagnosis time of *TraGraphRCA* by an average of 99.19% across both datasets. Furthermore, we observe that the diagnosis time of *TraGraphRCA* w/o bitmaps exhibits exponential growth, while the diagnosis time of *TraGraphRCA* with bitmaps shows linear growth. This demonstrates the improved scalability of *TraGraphRCA*.

5 Related Work

In the following, we provide a brief summary of existing approaches.

Trace-structure-based approaches. These methods typically analyze the structures of the traces corresponding to normal and abnormal requests separately, and localize the root cause based on their differences. TraceAnomaly [23] utilizes deep Bayesian networks to offline learn normal trace patterns and online detect and locate root causes. Pinpoint [15] aggregates traces to learn and dynamically update a normal behavior pattern of the application, and detects faults by comparing new requests with it. TraceCRL [47] utilizes contrastive learning and graph neural network methods to encode the structural and state information of each trace into vector and applies them to downstream analysis. FSF [29] leverages knowledge of failure propagation and the client-server model of communication to infer root causes. However, due to complex fault propagation patterns in systems, these methods cannot comprehensively analyze dependencies, especially lack the ability to deal with anomalies that propagate across traces.

Dependency-graph-based approaches. These methods typically start by mining the relationships among services to construct a dependency graph. They then traverse this graph to detect and recommend root causes. MicroScope [22] obtains network dependency through socket communication and recommends root causes with the similarity between anomalous nodes and frontend nodes. MicroRank [41] focus on latency issues and ranks root causes through combining PageRank and spectrum analysis. Sage [7] employs causal Bayesian networks to characterize the dependencies between microservices and uses graphical variational autoencoders to locate root causes. ImpactTracer [39] constructs an impact graph to describe fault propagation paths and utilizes a backward tracing algorithm to find root causes. However, these methods do not leverage the fine-grained information in traces, causing some issues which only affect specific requests escape from RCA.

Machine-learning-based approaches. These methods rely on historical or fault-injected labeled data. They construct a supervised model that determines root causes based on matching error representations from the historical data. MEPFL [50] and TFI [28] inject faults and collects fault traces in a test environment, and train a predictive model using supervised methods to locate root causes.

Putting *TraGraphRCA* in perspective. Compared to Machine-learning-based approaches, *TraGraphRCA* uses an unsupervised algorithm that does not require labeled data, making it an easily applicable algorithm in real-world microservice scenarios. *TraGraphRCA* combines the strengths of both trace-based approaches and graph-based approaches by finely mining the structural and status of traces at a microscopic level and analyzing the overall service dependencies at a macroscopic level. It provides a multi-level diagnostic report, achieving better diagnostic performance.

6 Threats to Validity

The threats to validity mainly come from the data quality used to construct the normal trace templates. If there are too many abnormal trace data included in the trace data during construction phase, it will result in the extraction of incorrect patterns and abnormal calculation results of statistical measures. This leads to incorrect matches during the diagnosis phase. On the other hand, if the trace data during the construction phase covers too few normal patterns, it will also result in the failure to correctly match the normal templates during the detection phase. Both of these situations can affect the accuracy of root cause analysis. During the actual deployment process, the normal patterns of *TraGraphRCA* is constructed incrementally, allowing the templates to be promptly updated to cover more normal patterns. Additionally, the trace data used for building

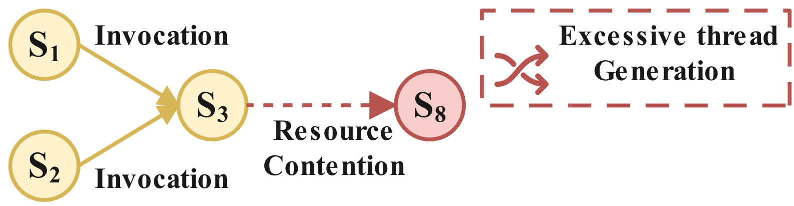


Fig. 10. The dependency relationship of a real-world case.

normal templates is preliminarily filtered based on trace indicators, aiming to exclude abnormal trace data as much as possible. These approaches help alleviate the aforementioned threats.

7 Practical Application

7.1 Overview

TraGraphRCA has been running in Huawei Cloud for 8 months and used to handle over 900 performance or reliability issues. Prior to using *TraGraphRCA*, SREs had to manually locate problems by reviewing alert panels and analyzing metric and trace information for large-scale interfaces. On average, it took them 3 hours to identify the root cause of a production environment failure. However, since implementing *TraGraphRCA*, the algorithm automatically provides multi-level root cause analysis results, helping SREs and developers narrow down the scope of investigation. Now, they can typically identify the root cause within 3 minutes. The accuracy of *TraGraphRCA*'s root cause analysis exceeds 80% in real-world business scenarios. SREs and developers have provided feedbacks that *TraGraphRCA*'s results have significantly improved their efficiency and saved manpower costs.

7.2 A Real-world Case

We introduce a real-world case to illustrate the root cause analysis process of *TraGraphRCA*.

In a microservices system that equipped with the *TraGraphRCA* tool, traces, events, and configuration files collected by agents during normal operation are sent to *TraGraphRCA*. *TraGraphRCA* dynamically constructs and updates multi-level trace templates and service dependency graph, persistently storing them. Users configure SLIs (Service Level Indicators) and other alert thresholds for their business applications. At a certain point, the system generates a significant number of alerts, triggering *TraGraphRCA* to perform root cause analysis.

During the diagnosis phase, *TraGraphRCA* collects traces and events, conducting in-depth trace-based analysis. It discovers traces passing through S_1, S_2, and S_3 (application services) experiencing significant latency delays, with some requests returning incorrect status codes. Simultaneously, it identifies numerous abnormal events in traces passing through S_8, indicating the generation of a large number of threads. Next, *TraGraphRCA* proceeds with graph-based analysis, revealing communication relationships between S_1, S_2, and S_3, as well as a non-communication relationship between S_3 and S_8. After ranking with PageRanker, *TraGraphRCA* recommends S_8 as the top-ranked root cause, pinpointing the abnormal events related to the excessive thread generation.

SREs quickly investigate S_8 based on the result of *TraGraphRCA* and confirm that the actual root cause was indeed the abnormal excessive thread generation in S_8. This led to the thread pool reaching its limit, causing anomalies in S_3, which also need to acquire threads from the thread pool, and further propagated the issue to S_1 and S_2 through their call relationships, as shown in Fig. 10.

In this example, *TraGraphRCA* successfully identified the genuine root causes, both at the service and event levels, significantly reducing the analysis time for SREs.

8 Conclusion

In this study, we present *TraGraphRCA*, a practical multi-level RCA approach that facilitates more detailed root cause reports for SREs. The core idea of *TraGraphRCA* is to combine both trace-structure-based and dependency-graph-based analysis to localize root causes at multi levels. To validate the effectiveness and efficiency of *TraGraphRCA*, we constructed two datasets, one from real production microservice systems and another from two widely-used microservices benchmarks namely TrainTicket and OnlineBoutique. Experimental results demonstrate that *TraGraphRCA* achieves significantly higher average top-1 accuracy (82.70%) compared to seven baseline methods. Moreover, *TraGraphRCA* has been deployed in Huawei Cloud for 8 months and achieves an accuracy of over 80% in root cause analysis.

References

[1] 2023. Kubernetes Homepage. http://kubernetes.io/. [Online].
[2] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks* 30 (1998), 107–117. http://www-db.stanford.edu/~backrub/google.html
[3] Donghun Cha and Younghan Kim. 2021. Service Mesh Based Distributed Tracing System. In *2021 International Conference on Information and Communication Technology Convergence (ICTC)*. 1464–1466. https://doi.org/10.1109/ICTC52510.2021.9620968
[4] Chaosblade. 2023. Chaosblade. https://github.com/chaosblade-io/chaosblade. Accessed Jan. 6, 2023.
[5] Yufu Chen, Meng Yan, Dan Yang, Xiaohong Zhang, and Ziliang Wang. 2022. Deep Attentive Anomaly Detection for Microservice Systems with Multimodal Time-Series Data. In *ICWS 2022*. IEEE, 373–378.
[6] FudanSELab. 2023. TrainTicket. https://github.com/FudanSELab/train-ticket. Accessed Jan. 6, 2023.
[7] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. 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* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 135–151. https://doi.org/10.1145/3445814.3446700
[8] GoogleCloudPlatform. 2023. OnlineBoutique. https://github.com/GoogleCloudPlatform/microservices-demo. Accessed Jan. 6, 2023.
[9] Grafana. 2023. Grafana Tempo. https://github.com/grafana/tempo. [Online].
[10] Zilong He, Pengfei Chen, Yu Luo, Qiuyu Yan, Hongyang Chen, Guangba Yu, and Fangyuan Li. 2023. Graph Based Incident Extraction and Diagnosis in Large-Scale Online Systems. In *ASE 2022*. ACM, Article 48, 13 pages.
[11] Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li, and Jing Han. 2021. Diagnosing Performance Issues in Microservices with Heterogeneous Data Source. In *ISPA/BDCloud/SocialCom/SustainCom, 2021*. IEEE, 493–500.
[12] Haiyu Huang, Xiaoyu Zhang, Pengfei Chen, Zilong He, Zhiming Chen, Guangba Yu, Hongyang Chen, and Chen Sun. 2024. TraStrainer: Adaptive Sampling for Distributed Traces with System Runtime State. *Proc. ACM Softw. Eng.* 1, FSE, Article 22 (July 2024), 21 pages. https://doi.org/10.1145/3643748
[13] Istio. 2023. Bookinfo. https://istio.io/latest/docs/examples/bookinfo/. Accessed: 2023/10/11.
[14] He Jiang, Jingxuan Zhang, Zhilei Ren, and Tao Zhang. 2017. An unsupervised approach for discovering relevant tutorial fragments for APIs. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 38–48.
[15] E. Kiciman and A. Fox. 2005. Detecting application-level failures in component-based Internet services. *IEEE Transactions on Neural Networks* 16, 5 (2005), 1027–1041. https://doi.org/10.1109/TNN.2005.853411
[16] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R. Lyu. 2023. Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-source Data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1750–1762. https://doi.org/10.1109/ICSE48619.2023.00150
[17] Rüdiger Lehmann. 2013. 3sigma-Rule for Outlier Detection from the Viewpoint of Geodetic Adjustment. *Journal of Surveying Engineering* 139 (11 2013), 157–165. https://doi.org/10.1061/(ASCE)SU.1943-5428.0000112
[18] Xing Li, Yan Chen, and Zhiqiang Lin. 2019. Towards automated inter-service authorization for microservice applications. In *SIGCOMM 2019*. ACM, 3–5.
[19] Xiaoyun Li, Guangba Yu, Pengfei Chen, Hongyang Chen, and Zhekang Chen. 2022. Going through the Life Cycle of Faults in Clouds: Guidelines on Fault Handling. In *ISSRE 2022*. IEEE, 121–132. https://doi.org/10.1109/ISSRE55969.2022.00022
[20] Yufeng Li, Guangba Yu, Pengfei Chen, Chuanfu Zhang, and Zibin Zheng. 2022. MicroSketch: Lightweight and Adaptive Sketch Based Performance Issue Detection and Localization in Microservice Systems. In *ICSOC 2022 (Lecture Notes in*

Computer Science, Vol. 13740). Springer, 219–236.

[21] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, Zhekang Chen, Wenchi Zhang, Xiaohui Nie, Kaixin Sui, and Dan Pei. 2021. Practical Root Cause Localization for Microservice Systems via Trace Analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. 1–10. https://doi.org/10.1109/IWQOS52092.2021.9521340

[22] Jinjin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments. In *ICSOC 2018*. Springer, 3–20.

[23] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. 2020. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 48–58. https://doi.org/10.1109/ISSRE5003.2020.00014

[24] Opentelemetry. 2023. Opentelemetry. https://opentelemetry.io. Accessed: 2023/7/14.

[25] Opentelemetry. 2023. OpenTelemetry Collector. https://github.com/open-telemetry/opentelemetry-collector. [Online].

[26] Opentelemetry. 2023. Opentelemetry span-events concept. https://opentelemetry.io/docs/concepts/signals/traces/#span-events. Accessed: 2023/7/14.

[27] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[28] Cuong Pham, Long Wang, Byung Chul Tak, Salman Baset, Chunqiang Tang, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. 2017. Failure Diagnosis for Distributed Systems Using Targeted Fault Injection. *IEEE Transactions on Parallel and Distributed Systems* 28, 2 (2017), 503–516. https://doi.org/10.1109/TPDS.2016.2575829

[29] Jesus Rios, Saurabh Jha, and Laura Shwartz. 2022. Localizing and Explaining Faults in Microservices Using Distributed Tracing. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. 489–499. https://doi.org/10.1109/CLOUD55607.2022.00072

[30] Carl Martin Rosenberg and Leon Moonen. 2020. Spectrum-Based Log Diagnosis. In *ESEM 2020*. ACM, 18:1–18:12.

[31] Sina Sheikholeslami, Moritz Meister, Tianze Wang, Amir H. Payberah, Vladimir Vlassov, and Jim Dowling. 2021. AutoAblation: Automated Parallel Ablation Studies for Deep Learning. In *Proceedings of the 1st Workshop on Machine Learning and Systems* (Online, United Kingdom) *(EuroMLSys '21)*. Association for Computing Machinery, New York, NY, USA, 55–61. https://doi.org/10.1145/3437984.3458834

[32] Yuri Shkuro. 2019. *Mastring Distributed Tracing*. Packt.

[33] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. Dapper, a large-scale distributed systems tracing infrastructure. (2010).

[34] Apache SkyWalking. 2023. Apache SkyWalking. https://skywalking.apache.org. Accessed July. 6, 2023.

[35] TraGraphRCA. 2023. TraGraphRCA implementation. https://anonymous.4open.science/r/TraGraphRCA-D16A/. Accessed Oct. 13, 2023.

[36] Ping Wang, Jingmin Xu, Meng Ma, Weilan Lin, Disheng Pan, Yuan Wang, and Pengfei Chen. 2018. CloudRanger: Root Cause Identification for Cloud Native Systems. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 492–502. https://doi.org/10.1109/CCGRID.2018.00076

[37] Li Wu, Johan Tordsson, Jasmin Bogatinovski, Erik Elmroth, and Odej Kao. 2021. MicroDiag: Fine-grained Performance Diagnosis for Microservice Systems. In *ICSE21 Workshop on Cloud Intelligence*. Madrid, Spain. https://inria.hal.science/hal-03155797

[38] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. 2020. MicroRCA: Root Cause Localization of Performance Issues in Microservices. In *NOMS 2020*. 1–9. https://doi.org/10.1109/NOMS47738.2020.9110353

[39] Ru Xie, Jing Yang, Jingying Li, and Liming Wang. 2023. ImpactTracer: Root Cause Localization in Microservices Based on Fault Propagation Modeling. In *DATE 2023*. 1–6. https://doi.org/10.23919/DATE56975.2023.10137078

[40] Zihao Ye, Pengfei Chen, and Guangba Yu. 2021. T-Rank:A Lightweight Spectrum based Fault Localization Approach for Microservice Systems. In *CCGrid 2021*. 416–425. https://doi.org/10.1109/CCGrid51090.2021.00051

[41] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. In *WWW 2021*. ACM, 3087–3098.

[42] Guangba Yu, Pengfei Chen, Pairui Li, Tianjun Weng, Haibing Zheng, Yuetang Deng, and Zibin Zheng. 2023. LogReducer: Identify and Reduce Log Hotspots in Kernel on the Fly. In *ICSE 2023*. 1763–1775.

[43] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable Fine-Grained Root Causes Analysis for Microservices on Multi-modal Observability Data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (San Francisco, CA, USA) *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 553–565. https://doi.org/10.1145/3611643.3616249

[44] Guangba Yu, Pengfei Chen, and Zibin Zheng. 2019. Microscaler: Automatic Scaling for Microservices with an Online Learning Approach. In *ICWS 2019*. IEEE, 68–75.

[45] Guangba Yu, Zicheng Huang, and Pengfei Chen. 2021. TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. *Journal of Software: Evolution and Process* (2021), e2413.

[46] Chenxi Zhang, Xin Peng, Chaofeng Sha, Ke Zhang, Zhenqing Fu, Xiya Wu, Qingwei Lin, and Dongmei Zhang. 2022. DeepTraLog: Trace-Log Combined Microservice Anomaly Detection through Graph-based Deep Learning. In *ICSE 2022*. IEEE, 623–634.

[47] Chenxi Zhang, Xin Peng, Tong Zhou, Chaofeng Sha, Zhenghui Yan, Yiru Chen, and Hong Yang. 2022. TraceCRL: Contrastive Representation Learning for Microservice Trace Analysis. In *ESEC/FSE 2022*. ACM, 1221–1232.

[48] Hao Zhou, Ming Chen, Qian Lin, Yong Wang, Xiaobin She, Sifan Liu, Rui Gu, Beng Chin Ooi, and Junfeng Yang. 2018. Overload Control for Scaling WeChat Microservices. In *SoCC 2018*. ACM, 149–161.

[49] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2021. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. *IEEE TSE* 47, 2 (2021), 243–260.

[50] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs *(ESEC/FSE 2019)*. Association for Computing Machinery, 683–694.