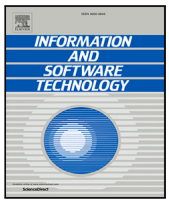
Information and Software Technology 189 (2026) 107938

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



MHP-RCA: Multivariate Hawkes Process-based Root Cause Analysis in microservice systems

Zekun Zhang [a], Jian Wang [a,*], Bing Li [a], Yu Liu [a], Hongyue Wu [b], Patrick C.K. Hung [c]

[a] School of Computer Science, Wuhan University, China
[b] College of Intelligence and Computing, Tianjin University, China
[c] Faculty of Business and Information Technology, Ontario Tech University, Canada

ARTICLE INFO

Keywords:
Microservice
Root cause analysis
Audit log
Metric
Service-based system

ABSTRACT

Context: Recent years have witnessed a prevailing trend of developing applications using microservice architectures. Microservice systems typically involve multiple containers that share resources on a single physical host, thereby complicating the interdependencies among microservices. This complexity significantly hinders the identification of root causes of performance issues.
Objective: Performance issues can manifest in various forms. Existing approaches often overlook other potential failure indicators, such as process anomalies that are discernible in audit logs. This paper aims to refine the granularity of root cause analysis to the process level.
Methods: This paper proposes a novel approach called MHP-RCA (Multivariate Hawkes Process-based Root Cause Analysis), which integrates diverse data types, including metrics and audit logs, to localize the root cause in microservice systems. MHP-RCA generates anomalous events from the observable data, then leverages the multivariate Hawkes process to construct causal graphs for effective root cause identification.
Results: Extensive experiments, involving the injection of various anomalies into four widely used open-source benchmarks, demonstrate that MHP-RCA surpasses multiple baseline methods in most cases. Compared to the best-performing baseline approach, MHP-RCA achieves an average overall improvement of 2.5% in AC@1 and 3.7% in AC@5.
Conclusion: The proposed method MHP-RCA, which considers audit logs and metrics, can localize the root cause of microservice anomalies at the process level.

1. Introduction

In recent years, the rapid evolution of the Internet has necessitated fast development and maintenance cycles for software applications. Traditional monolithic architectures, characterized by their tightly integrated business logic, can no longer meet the requirements of agile developments. With advancements in cloud computing, containerization, and DevOps (Development and Operations), microservice architecture [1] has emerged as the predominant model for constructing large-scale applications. This architecture involves breaking down complex applications into a collection of small, autonomous services, enabling more flexible development and deployment. In large-scale microservice systems, millions of performance alerts can be generated daily, covering issues such as system failures, performance degradation, and other abnormal situations. Monitoring tools have been widely used to observe the performance of microservices, aiding operators in the detection of anomalies [2,3]. However, besides receiving alerts from monitoring tools, operators must be aware of the anomaly's location (e.g., the faulty service instance) and its cause (e.g., CPU overload). Traditionally, operators analyze all pertinent alerts and network information to achieve fault recovery [4], which is slow and time-consuming. Factors including complex operational environments, dynamic virtual machine configurations, and numerous service nodes further complicate the root cause localization task, which presents following challenges:

The anomaly propagation relationships between microservices are complex. In microservice systems, multiple containers may be deployed on the same physical node, sharing underlying resources like CPU, storage, and network bandwidth. This shared nature complicates the interdependencies among containers. For example, a defect in one container can result in the excessive use of shared network bandwidth, adversely impacting the regular operation of other containers. This type of error propagation is not represented in traditional invocation graphs

* Corresponding author.
E-mail address: jianwang@whu.edu.cn (J. Wang).

https://doi.org/10.1016/j.infsof.2025.107938
Received 12 August 2024; Received in revised form 26 January 2025; Accepted 20 October 2025
Available online 24 October 2025

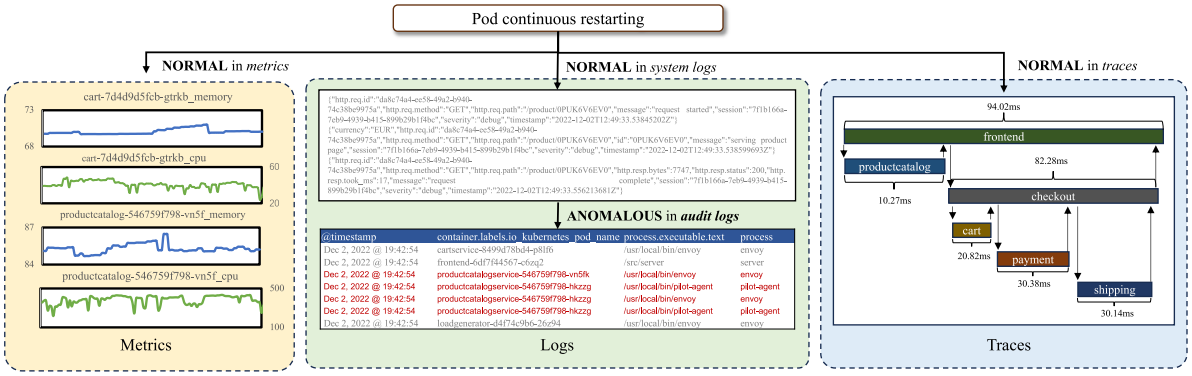


Fig. 1. Illustration of the necessity of audit logs in a specific anomaly scenario.

because it involves competition and sharing of underlying resources, rather than code dependencies. Additionally, not every microservice system can obtain a complete service call graph through distributed tracing technology. In this context, a data-driven causal graph, as an alternative to the precise invocation graph, can capture these propagation patterns. The central concept is to infer causal relationships from origin data and identify the root causes that are most likely to trigger failures.

Anomalies are diverse and involve various observable data types. Microservices systems, due to their distributed architecture, exhibit complex and diverse fault patterns, posing challenges for fault localization. Fault modes in microservices systems can be roughly classified into hardware failures, software errors, and network problems [5]. These faults typically manifest through different modalities of observable data, such as metrics (typically indicating hardware or network issues), logs (typically revealing software errors), and traces (often capturing network issues). Relying solely on a single data modality is insufficient for comprehensive fault localization; instead, integrating multiple modalities yields a more holistic view and thus aids in more accurate problem identification. Some existing studies [6–8] identify root causes by constructing a service dependency graph (SDG) based on monitored traces. However, because trace collection requires costly code instrumentation, metrics and logs are more commonly found in microservices systems. Consequently, in this paper, we adopt these two data modalities to focus on several common fault scenarios in microservice systems, such as resource issues, network delays, and pod anomalies, and further explores how integrating multi-modality data can enhance accuracy and efficiency in fault localization.

Traditional observable data have limited the effectiveness for process-level root cause localization. We argue that relying solely on a single data source for anomaly detection and fault diagnosis is insufficient. As illustrated in Fig. 1, we demonstrate the use of various data types to identify the "continuous pod restarting" anomaly. Specifically, metrics fail to clearly reveal the anomaly due to their fixed interval recording, which fails to detect this kind of process-level issue. System logs and traces may also not report anomalies during such events. However, as shown in Fig. 1, audit logs successfully document the anomaly, which reports the continuous restart process of the "product catalog service" with numerous logs of anomaly processes.

To address the above issues, we propose MHP-RCA (Multivariate Hawkes Process-based Root Cause Analysis), a root cause analysis approach based on the Hawkes process [9]. Firstly, data from various sources in microservice systems exhibit significant format disparities. Logs are recorded randomly after events occur, while metrics are typically collected at fixed time intervals. To align these divergent data types, we characterize system failures using anomalous events. Secondly, upon the occurrence of an anomalous event, MHP-RCA constructs the causal graph using the Hawkes process. Finally, we conduct root cause localization based on the causal graph constructed and identify the most likely root causes. The main contributions of this paper are outlined as follows:

1. We propose a root cause analysis method at the process level. By combining metrics and audit logs, it can detect resource-level failures and process-level issues like pod restarts and abnormal scaling. The data and source code have been open-sourced.[1]
2. We propose a causal graph model for failure propagation in microservice systems. This model represents anomalies in multi-modal data as point events and constructs a failure propagation graph using the Hawkes process-based Granger causality.
3. We extensively evaluate MHP-RCA on four real-world datasets. Experimental results have shown that MHP-RCA outperforms the state-of-the-art methods over multiple metrics.

The subsequent part of this study is organized as follows: Section 2 presents the foundational knowledge of multivariate Hawkes processes and audit logs. Section 3 provides detailed explanations of MHP-RCA. Section 4 outlines the evaluation metrics and experimental procedures employed to validate the proposed method. Section 5, reviews related work and provides an overview of the current research of root cause localization in microservices systems. Finally, Section 6 draws conclusions and discusses future work based on the findings presented in this study.

2. Background

2.1. Multivariate Hawkes process

The multivariate Hawkes process is a statistical method frequently employed for modeling time series data, demonstrating the mutual relationships between events. It is usually depicted as a collection of correlated intensity functions. Each intensity function is utilized to model the mutual triggering mechanisms. These functions are influenced by historical event sequences as well as other event sequences, enabling the model to capture both excitatory and inhibitory relationships between these sequences. The intensity functions are defined as follows:

$$\lambda_v(t) = \mu_v + \sum_{v' \in \boldsymbol{V}} \int_{t' \in \boldsymbol{T}_{t^-}} \alpha_{v',v} \kappa(t - t') dC_{v'}(t'), \tag{1}$$

where $\boldsymbol{T}_{t^-} = \{t' \mid t' \in \boldsymbol{T}, t' < t\}$, the intensity function $\lambda_v(t)$ is determined by the basic intensity μ_v and excitation intensity $\sum_{v' \in \boldsymbol{V}} \int_{t' \in \boldsymbol{T}_{t^-}} \alpha_{v',v} \kappa\left(t - t'\right) dC_{v'}\left(t'\right)$. The basic intensity μ_v denotes the probability of spontaneous occurrence of an event, and the excitation intensity is used to model the effect of other sequences on the current event [10]. $\alpha_{v',v}$ denotes the influence function of historical events v' on subsequent events v. $\kappa\left(t - t'\right)$ is the decay function, which denotes the decay of $\alpha_{v',v}$ over time.

[1] https://github.com/WHU-AISE/MHP-RCA

Table 1
An example of microservice audit logs.

Timestamp	Process.name	Process.args	Namespace	Version	App (Microservice)
2023/6/22 3:32	containerd-shim	/usr/bin/containerd-shim-runc-v2, -namespace, moby, -id, de0fd0e969634e538d3897, -address, /var/run/desktop-containerd/containerd.sock	default	v2	reviews
2023/6/22 3:32	runc:[2:INIT]	runc, init	default	v2	reviews
2023/6/22 3:32	runc	runc, –version	default	v1	details
2023/6/22 3:31	bridge	/var/lib/cni-plugins/bin/bridge	default	v1	ratings

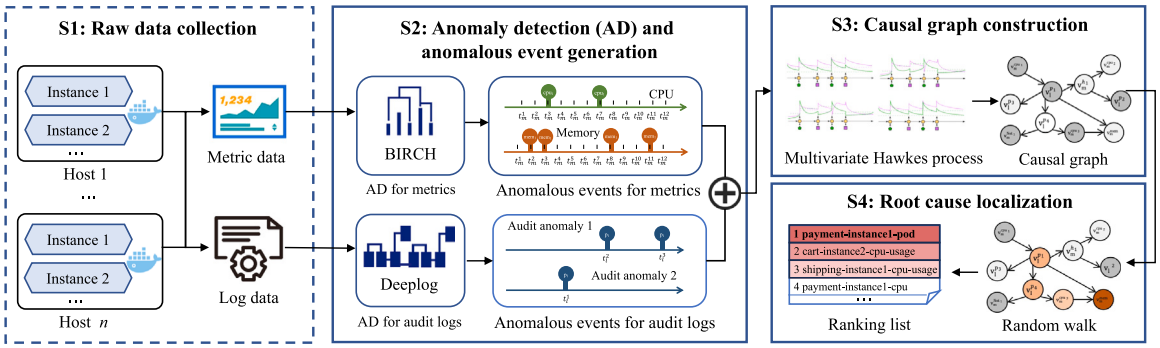


Fig. 2. Framework of MHP-RCA.

2.2. Audit logs

Audit logs provide detailed records of operations, events, and behaviors in a system. They are commonly used for tracking security measures related to system activities. Traditional audit logs provide detailed records of key events such as user logins, file accesses, permission changes, and network communications. In addition to the aforementioned information, audit logs in microservices systems also capture internal behaviors such as changes in service status, fault transfer events, and other operations. Table 1 displays audit logs of a microservice system, including information like timestamps, process names, process parameters, namespaces, and app versions. By analyzing audit logs, it is possible to identify process-level operations that deviate from normal behaviors. An illustrative example of this would be the continuous restarting of pods, which can be identified through careful examination of these detailed logs.

3. Approach

3.1. Overall framework

As depicted in Fig. 2, the framework of MHP-RCA consists of four primary components: (*i*) Raw data collection; (*ii*) Anomaly detection and anomalous event generation; (*iii*) Causal graph construction; and (*iv*) Root cause localization.

The initial stage systematically gathers metrics and audit logs based on a predefined criteria. Then we implement the anomaly detection to generate anomalous events, including metrics conspicuously deviating from normal ranges or abnormal patterns in audit logs. During the causal graph construction, we analyze historical anomalous events to infer causal relationships among variables and simulate the fault propagation to build the causal graph. This stage leverages a hybrid approach combining the multivariate Hawkes process with Granger causality [11]. In the last stage, we rank anomalous events to identify the most likely root cause using the personalized PageRank algorithm. The higher an event is ranked, the greater its likelihood of being the root cause. This is critical for quickly restoring services in real-world scenarios, as a higher success rate for the top-ranked solution leads to shorter troubleshooting times. Each component of MHP-RCA is described as follows.

3.2. Raw data collection

Microservice systems generate various metrics during runtime, which are collected at fixed intervals. These metrics include: (i) *Application performance metrics* reflect the overall operational state of the microservice system, such as the error rate and response time [12]; (ii) *Resource utilization metrics* reflect the observed resource usage at the physical or virtual layer, such as CPU, memory, and I/O usage of hosts [4]; and (iii) *Container metrics* reflect container-level information, such as container resource utilization and the number of microservice instances. These metrics appear in different types: for instance, application performance is usually represented by histogram metrics analyzing sample distributions, while resource utilization is often depicted as gauge metrics reflecting the system's current state. We formulate these metrics as time series vectors:

$$S(t) = \left[m_1^t, m_2^t, \ \ldots, m_n^t\right], \tag{2}$$

where m_i^t indicates the value of metric i at time t, and n is the total number of metrics.

Considering that code modifications might be necessary to insert tracing context in microservice systems, we did not collect traces during the data collection process, but instead, only collected metric and log data.

3.3. Anomaly detection and data alignment

Given the limited recording capability and storage capacity, high-precision data recording often becomes prohibitively expensive. Considering these cost constraints [13], low-precision discrete event sequences are established in practical scenarios. In identifying root causes in microservices, our primary focus is on anomalous events, which are crucial indicators of system performance issues.

3.3.1. Anomaly detection and anomalous event generation

As for logs, these anomalous events indicate abnormal logs. As for metrics, anomalous events represent deviations from normal data points within a specific time frame. For example, if the memory usage of a specific container reveals a sudden value that significantly deviates from other data points, this particular metric is deemed an anomaly.

We observe that the metric data exhibits periodicity. To address this, we apply the SR algorithm [14] to remove seasonal effects, ensuring

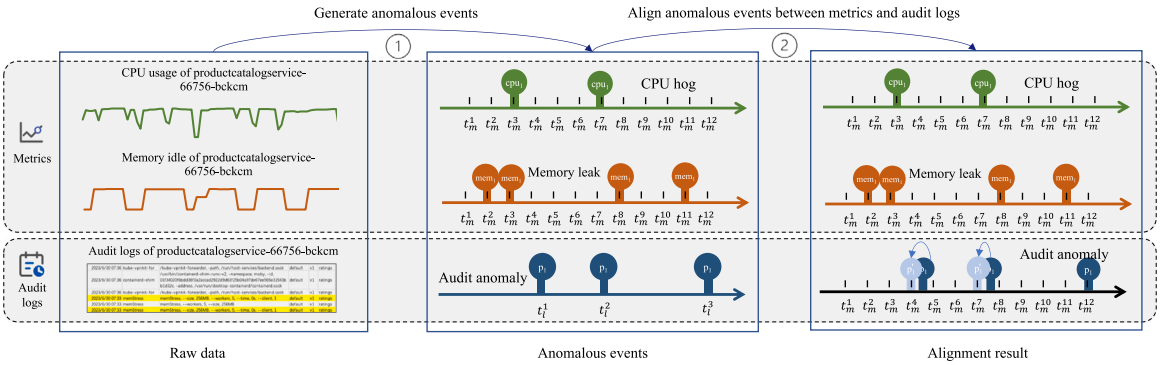


Fig. 3. Alignment of metrics and audit logs.

that the processed data is suitable for subsequent causal inference. Due to the differences in anomaly detection granularity across microservice systems, we adopt a unified anomaly detection method in MHP-RCA. Specifically, anomaly events are extracted by clustering the metric data using the BIRCH clustering algorithm [15]. The BIRCH clustering is an unsupervised learning method that can adapt to different metrics without relying on predefined rules or labels. For each metric k, the historical data within the time window T is denoted as $m_k = \{x_k^1, x_k^2, \ldots, x_k^T\}$, which is input into the clustering algorithm. The algorithm divides the data into h clusters, with a time complexity of $O(N)$. Each data point x_k^i is assigned to a cluster $c_k^i \in \{1, \ldots, h\}$, where c_k^i represents the cluster label of the data point. If the clustering result consists of only one cluster (i.e., $h = 1$), all data points are labeled as 0, indicating no anomaly. If the clustering result contains two or more clusters (i.e., $h \geq 2$), the data is considered to have anomalies, and data points assigned to non-major clusters are marked as anomalies, with $c_k^i = 1$. Therefore, the anomaly event vector v_k^m is defined as:

$$v_k^m = \{c_k^1, c_k^2, \ldots, c_k^T\}, \tag{3}$$

where $c_k^i = 0$ indicates normal data, and $c_k^i = 1$ indicates anomalous data. The set of anomalous events for all metrics is formed by combining all v_k^m as V_m.

For audit logs, we employ the Deeplog [16] to identify log anomalies. We also record the time of a specific anomalous event, such as those starting with '*Error*' or '*Runc*', marking the anomaly value at that moment as 1. Then the original anomalous event can be represented as:

$$V_l = \{c_l^1, c_l^2, \ldots, c_l^{n_a}\}, \tag{4}$$

where n_a represents the number of anomalous events for audit logs.

3.3.2. Alignment process for metrics and audit logs

The formats of logs and metrics differ significantly. Specifically, logs are in text format with randomly timed entries, while metrics consist of numerical data collected at a fixed time interval. To apply the multivariate Hawkes process, it is necessary to represent events as a unified time series. However, the heterogeneity of audit logs and metrics introduces a challenge: audit logs are recorded as discrete events with irregular timestamps, while metrics are collected periodically at fixed intervals. To address this discrepancy, we align the timestamps of these two data sources.

Specifically, for anomalous events of audit logs, the timestamps are denoted as $T_l = \{t_l^1, t_l^2, t_l^3, \ldots, t_l^{n_a}\}$. For anomalous events of metrics, the timestamps are sampled periodically, represented as $T_m = \{t_m^1, t_m^2, t_m^3, \ldots, t_m^T\}$. To align audit logs with metrics, we define an indicator function $I(t_m^j, T_l)$ as follows:

$$I(t_m^j, T_l) = \begin{cases} 1, & \text{if } \exists t_l^i \in [t_m^j, t_m^{j+1}), \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Then we generate an aligned sequence for audit logs:

$$V_l' = \{I(t_m^1, T_l), I(t_m^2, T_l), \ldots, I(t_m^{n_1}, T_l)\}, \tag{6}$$

where V_l' represents whether each time interval in T_m overlaps with any anomalous event in the audit logs. The alignment process is illustrated in Fig. 3, providing an intuitive view of how heterogeneous data sources are integrated for Hawkes process modeling. This alignment ensures that the lengths of the metric and audit log sequences are identical, enabling the construction of a unified event set [17]:

$$\boldsymbol{V} = V_l' \cup V_m, \tag{7}$$

where V_m is the anomalous events of metrics and V_l' is the anomalous events of audit logs after alignments.

It is worth noting that while this alignment process may introduce minor granularity loss in audit logs, this is mitigated by setting a sufficiently high sampling rate for metrics. The unified representation facilitates the construction of a consistent causal graph, capturing the interrelation between metrics and audit logs effectively.

3.4. Causal graph construction based on the Hawkes process

In real-world scenarios, microservices are typically deployed in a distributed system. There is a cascading effect of anomalies [6], where services with invocation relationships or shared resources are easily affected. As a result, multiple sets of metric or log anomalies may appear when an alert occurs. In such cases, constructing a causal graph becomes a crucial approach to simulate fault propagation paths, especially when precise topological information is unavailable. In this paper, we analyze historical anomalous events to establish causal relationships between variables. We assume that the causal graph remains static within the observation window. This means that while the metrics and audit logs evolve over time, the structure of the causal relationships between the variables, as inferred from these data, does not change during the observation period. Then, a hybrid method that combines the multivariate Hawkes process with Granger causality is proposed.

3.4.1. The Granger causality in the Hawkes process

Unlike traditional Granger causality tests [11] that require stationary input data, the Hawkes process effectively captures temporal dependencies and self-excitation characteristics inherent in non-stationary time series. In our approach, we rely on the Hawkes process's flexibility to dynamically adjust to changing event intensities, thereby ensuring that the model can appropriately handle non-stationary behaviors in the microservices system. Specifically, the multivariate Hawkes process is combined with Granger causality to infer causal relationships. Assuming $A = \{a_1, a_2, \ldots, a_t, \ldots\}$ and $B = \{b_1, b_2, \ldots, b_t, \ldots\}$ are random processes, if the historical values of A can statistically predict the current value of B, it implies that there is a causal relationship between A and B. For example, insufficient memory on a host machine (A') resulting in increased response time of microservice m_1 (B') can be

represented as $A' \to B'$. It is noted that we exclude the consideration of instantaneous causal directions, which are challenging to infer and distinguish, particularly within the Granger causality framework. In our model, anomalous events are conceptualized as a multivariate point process, allowing the Granger causal structure among these events to be depicted as $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ and $\mathbf{E}$ denote the set of anomalous events and causal edges, respectively.

In the context of a multivariate point process, the generative process follows the Hawkes process intensity functions. The Granger causal structure $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ is established if and only if $\phi_{v'}(t) = 0$ is satisfied for all $t \in T$. We can further conclude that in the multivariate Hawkes process model, v' is not a Granger cause of v if and only if v and v' are independent, i.e., $\alpha_{v',v} = 0$. Therefore, in the presence of a potential causal structure $\mathcal{G}_{\mathcal{V}} = (\boldsymbol{V}, \boldsymbol{E})$, the intensity function of the Hawkes process can be further simplified as:

$$\lambda_v(t) = \mu_v + \sum_{v' \in \boldsymbol{PA}_v} \int_{t' \in \boldsymbol{T}_{t-}} \alpha_{v',v} \kappa(t - t') dC_{v'}(t'), \tag{8}$$

where v denotes the anomalous event, and $\boldsymbol{PA_v}$ represents the set of Granger cause events of v under the given causal structure $\mathcal{G}_{\mathcal{V}} = (\boldsymbol{V}, \boldsymbol{E})$, i.e., the set of parent nodes for $\mathcal{G}_{\mathcal{V}}$ in the causal graph. This intensity function is adapted to handle non-stationary characteristics by allowing the effects of past events to decay over time at a rate determined by the model parameters. As a result, even if the event sequences exhibit non-stationary properties, such as trends or periodic fluctuations, the model can still effectively capture the causal relationships between events.

In real-world environments, metrics in microservice systems are often collected at regular intervals. For instance, we collect metrics every five seconds, which reduces granularity but balances localization time and accuracy. Hence, it is necessary to model the event sequence in a discrete-time domain. As a result, the original continuous time domain $\boldsymbol{T}$ becomes a discrete time domain $\boldsymbol{T} = \{0, \Delta t, 2\Delta t, \ldots\}$, and $\boldsymbol{T}_{t^-} = \{t' \mid t' \in \boldsymbol{T}, t' \leq t - \Delta t\}$. Thus the event sequence $C = \{C_v(t) \mid t \in \boldsymbol{T}, v \in \boldsymbol{V}\}$ can be equivalently represented by the time sequence $\boldsymbol{X} = \{X_{v,t} \mid v \in \boldsymbol{V}, t \in \boldsymbol{T}\}$, where $X_{v,t} = C_v(t + \Delta t) - C_v(t)$ denotes the number of occurrences of the anomalous events in the time interval $[t, t+\Delta t)$. In this case, we give an extension of the Hawkes process model in the discrete-time domain, with the representation of its intensity function as follows:

$$\lambda_v(t) = \mu_v + \sum_{v' \in PA_v} \sum_{t' \in T_{t-}} \alpha_{v',v} \kappa(t - t') X_{v',t'}. \tag{9}$$

where μ_v represents the base intensity of node v, $\alpha_{v',v}$ denotes the causal influence of parent node v' on v, and $\kappa(t - t')$ is the time decay kernel function that describes the temporal dependency between events. If $\kappa(t - t')$ is an exponential decay function $\exp(-\beta(t - t'))$, the further away the occurrence time of v' is from the current time point t, the smaller its influence becomes. When Δt is sufficiently small, the Poisson distribution can effectively approximate the event occurrence probability, enabling the model to capture event relationships in the discrete-time domain.

3.4.2. Bayesian scoring criterion in discrete processes

In the multivariate Hawkes process-based Granger inference, we construct the likelihood function based on the Bayesian Information Criterion (BIC) [18]. The objective is to find parameters that maximize the probability of modeling anomalous events by causal inferences. By optimizing parameters, we can capture the causal relationships between abnormal events. This causal relationship reflects the path of anomaly propagation. To integrate event sequences of the discrete-time domain into our framework, we construct the likelihood function based on the Poisson distribution. Meanwhile, to establish a reasonable causality structure in discrete states, we introduce the l_0 norm as a sparse penalty term. This inclusion is crucial for ensuring the sparsity of the causality structure. The final likelihood function is expressed as a function of $\mathcal{G}$, Θ, and $\boldsymbol{X}$, as shown in Eq. (10):

$$\begin{aligned} L(\mathcal{G}, \Theta; \boldsymbol{X}) &= \sum_{v \in \mathbf{V}} \sum_{t \in \mathbf{T}} P\left(X_{v,t} \mid H_t^{\mathbf{PA}_v}\right) \\ &= \sum_{v \in \mathbf{V}} \sum_{t \in \mathbf{T}} \left[-\lambda_v(t)\Delta t + X_{v,t} \log\left(\lambda_v(t)\right)\right] \\ &\quad + \underbrace{\sum_{v \in \mathbf{V}} \sum_{t \in \mathbf{T}} \left[-\log\left(X_{v,t}!\right) + X_{v,t} \log(\Delta t)\right]}_{\text{Const}}, \end{aligned} \tag{10}$$

where $\Theta = \left\{\mu_v \mid v \in \mathbf{V}\right\} \cup \left\{\alpha_{v',v} \mid (v', v) \in \mathbf{E}\right\}$ represents the model parameter. $H_t^{\mathbf{V}} = \{(t_i, v_i) \mid t_i, v_i \in \mathbf{V}\}$ denotes the set of events occurring before moment t in the cause event type of v. We set the BIC penalty term $l_0 = \frac{p \log(m)}{2}$ into the likelihood function, and the updated likelihood function is defined as follows:

$$L(\mathcal{G}, \Theta; \boldsymbol{X}) - \frac{p \log(N)}{2}, \tag{11}$$

where $p = |\mathbf{V}| + |\mathbf{E}|$ represents the l_0 norm, indicating the total number of nodes and edges in the causal graph. This term constrains the complexity of the graph to prevent overfitting. By introducing the penalty term, unnecessary nodes and edges are reduced, resulting in a sparse causal graph structure that facilitates easier interpretation of causal relationships. The weight of the regularization term $\log(N)/2$ can be adjusted according to the number of event samples, enabling greater adaptability.

Once the likelihood function $L(\mathcal{G}, \Theta; \boldsymbol{X})$ is determined, we search for parameters that maximize the probability of causality. For the given hypothetical causal structure $\mathcal{G}$, we estimate Θ using the EM optimization algorithm [19], where the iterative optimization formula is derived as:

$$\mu_v^{(i)} = \frac{\Sigma_{t \in \mathbf{T}} q_{v,t}^{\mu} X_{v,t}}{|\mathbf{T}| \Delta t}, \tag{12}$$

and the iterative formula [20] for the parameter $\alpha_{v',v}$ is

$$\alpha_{v',v}^{(i)} = \frac{\sum_{t \in T} q_{v,t}^{\alpha}\left(v', t'\right) X_{v,t}}{\sum_{t \in T} \sum_{t' \in T_{t-}} \kappa\left(t - t'\right) X_{v',t'} \Delta t}, \tag{13}$$

here

$$q_{v,t}^{\mu} = \frac{\mu_v^{i-1}}{\lambda_V^{i-1} t}, \tag{14}$$

$$q_{v,t}^{\alpha}(v', t') = \frac{\alpha_{v',v}^{i-1} \kappa(t - t') X_{v',t'} \Delta t}{\lambda_V^{i-1} t}, \tag{15}$$

$$\lambda_v^{(i-1)}(t) = \mu_v^{(i-1)} + \sum_{v' \in \mathbf{PA}_v} \sum_{t' \in \mathbf{T}_{t-}} \alpha_{v',v}^{(i-1)} \kappa(t - t') X_{v',t'}. \tag{16}$$

The EM algorithm iteratively alternates between the expectation step, where weights $q_{v,t}^{\mu}$ and $q_{v,t}^{\alpha}$ are updated, and the maximization step, where μ_v and $\alpha_{v',v}$ are re-estimated based on the observed data. This iterative process continues until convergence criteria are met:

$$\max\left(\left|\mu_v^{(i)} - \mu_v^{(i-1)}\right|, \left|\alpha_{v',v}^{(i)} - \alpha_{v',v}^{(i-1)}\right|\right) < \epsilon, \tag{17}$$

where ϵ is the convergence threshold, typically set to a small positive value (e.g., 10^{-4}). To avoid infinite loops, a maximum number of iterations $I_{\max}$ can be set. When the number of iterations reaches $I_{\max}$, the iteration terminates. This can be expressed as:

$$i \geq I_{\max}. \tag{18}$$

The decay kernel $\kappa(t - t')$ ensures that the temporal influence of past events is appropriately weighted, allowing the model to capture time-sensitive causal relationships effectively. We use the above formulas to iteratively update the model parameter Θ of the given structure $\mathcal{G}$ until convergence. We obtain a candidate set $S(\mathcal{G}')$ of causal structure $\mathcal{G}'$ by adding, deleting, and reversing edges for anomalous events. Based on the converged Θ, we calculate the BIC score of all $\mathcal{G}'$ in $S(\mathcal{G}')$. The

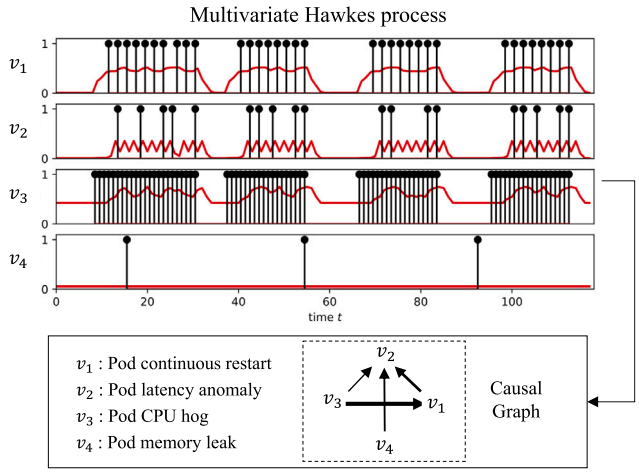


Fig. 4. Example of the causal graph built by the multivariate Hawkes process.

final causal structure $\mathcal{G}^*$ with the largest BIC score optimally represents the causal graph among anomalous events. Fig. 4 illustrates the construction of a causal graph. In this figure, $v_1 - v_4$ denotes distinct event types, black dots indicate anomalous events and red curves represent the intensity functions of the multivariate Hawkes process. By utilizing the multivariate Hawkes process, we can construct the causal graph among anomalous events in a microservice system, as shown in the dashed box.

3.5. Root cause localization

Graph-based random walk algorithms, notably the PageRank [21], have been used in root cause localization tasks for microservice systems [7,12]. These algorithms typically involve transition probabilities of moving from one node to another in a graph. We apply the Personalized PageRank (PPR) algorithm to rank anomalous event types in the constructed causal graph. In this way, the probability of transition will be more inclined to follow a causal relationship, modeling the path of fault propagation. This random walk process is computed as follows:

$$r(\mathcal{G}^*) = (1 - c) \cdot \mathrm{M}(\mathcal{G}^*)r + cu(\mathcal{G}^*), \tag{19}$$

where the PPR vector $u(\mathcal{G}^*)$ represents the BIC score of $\mathcal{G}^*$, nodes with higher BIC scores have higher probabilities of being visited in the random walk process. $\mathrm{M}(\mathcal{G}^*)$ represents the matrix of the causal graph. The random walk pointer c is a constant ranging from $(0, 1)$. Previous studies [21] have shown that c has a minor impact on the final ranking results. In this paper, we take a default value of 0.15 [22]. The variable $r(\mathcal{G}^*)$ represents the PageRank value of each anomalous event type. By sorting these PageRank values, we can obtain the final list of root causes, e.g., the ranking of anomalous event types.

4. Experiments

We evaluate MHP-RCA with several baseline methods and answer the following research questions:

- RQ1: Is MHP-RCA more effective compared to state-of-the-art causal inference-based methods?
- RQ2: Does MHP-RCA perform well across various anomaly cases?
- RQ3: How does the efficiency of MHP-RCA compare with other methods?
- RQ4: What is the contribution of each component in MHP-RCA?
- RQ5: What is the impact of various parameters on root cause analysis results?
- RQ6: How robust is MHP-RCA when handling incomplete data?

4.1. Experiment preparations

4.1.1. Implementation details

We utilize a Kubernetes cluster comprising one master node and two worker nodes for microservice deployments. Metric and log collection tools are deployed on the master node, while the benchmark systems are deployed on worker nodes. With the Filebeat[2] tool, system logs are aggregated and continuously collected over specified time windows. Audit logs are collected using the AuditBeat,[3] a tool provided by Elasticsearch, which captures process-level events of containers via the '*auditd*' module. And we use the Prometheus and Node exporter[4] to collect metrics. The detailed experimental environment is outlined in Table 2.

Then we list the implementation details of MHP-RCA. In the anomaly detection phase, anomalous events of metrics and logs are generated through the BIRCH clustering with the initial diameter set to 100. Then the anomalous events are treated as point events, which can be modeled using the Hawkes process. Initial parameters of Hawkes process are set as $k = 0.5$, $\alpha = 1$, $\mu = 0.01$, and $\beta = 0.1$. Then, the Granger causality is used to infer the fault propagation graph, and the causal graph is optimized using the EM algorithm, with the convergence threshold ϵ set to 10^{-6}, and the maximum number of iterations $I_{\max}$ set to 500. Finally, the root cause ranking list is generated through the random walk (Personalized PageRank) on the causal graph.

4.1.2. Benchmarks

Experiments were conducted on four benchmark systems, where 120 faults were injected. The detailed information about each benchmark is outlined below.

- **Bookinfo**[5] (dataset $\mathcal{A}$) is a microservice application showing the capabilities of the Istio service mesh. It consists of four microservices, each representing a functional aspect of an online bookstore.
- **Online-Boutique**[6] (dataset $\mathcal{B}$) is a cloud-native online shopping system consisting of 11 microservices. Online-Boutique offers the ability to browse and purchase products via its e-commerce platform. Notably, among these 11 microservices, three are simulation-based and do not engage in actual business operations.
- **Sock-Shop**[7] (dataset $\mathcal{C}$) is a cloud-native system that comprises seven business microservices and four database microservices. It simulates an e-commerce website that sells socks and is widely used for testing and experimentation in cloud-native environments.
- **Train Ticket**[8] (dataset $\mathcal{D}$) is a microservice-based reservation system for train tickets, covering core functionalities such as user authentication, order management, and payment. The system consists of 42 microservices communicating via gRPC protocol, with 27 instances directly involved in business processing.

4.1.3. Fault injections

To gather data representative of real-world business scenarios, we use Locust,[9] a widely used tool for simulating concurrent user requests. Recognizing that different microservices experience varying request rates in actual business settings, we tailor the simulated request rate accordingly. In our experiments, we collected data from the initial

[2] https://www.elastic.co/beats/filebeat
[3] https://www.elastic.co/beats/auditbeat
[4] https://github.com/prometheus/node_exporter
[5] https://istio.io/latest/docs/examples/bookinfo
[6] https://github.com/GoogleCloudPlatform/microservices-demo
[7] https://github.com/microservices-demo/microservices-demo
[8] https://github.com/FudanSELab/train-ticket
[9] https://locust.io

Table 2
The detailed experimental environment.

Hardware	CPU: 8-core 2.20 GHz, RAM: 32GB, OS: Ubuntu
Container Env	Kubernetes 11.3.1, Docker Client 7.31, Docker Server 7.31
Data Collection	Metrics: Istio 1.4.5, Prometheus 6.3, NodeExporter 1.41. Audit logs: Elastic Search 8.4.1, Kibana 8.4.1, AuditBeat 8.4.1
Benchmark System	Bookinfo 1.17.0, Online-Boutique 0.8.0, Sock-Shop 0.4.8, Train-Ticket 1.0.0

10 min and generated anomalous events at 5-second intervals (Δt=5s). This approach resulted in a dataset comprising 120 data points for each metric. To simulate potential failures akin to those in real-world scenarios, we inject various types of anomalies, such as container-level failures, latency delays, and resource limitations. The detailed methods used for injecting these anomalies are as follows:

- **Continuous pod restarting**. Microservices systems may face continuous pod restarting due to reasons such as insufficient resources, failed image pulling, or internal program errors, leading to temporary service unavailability. We utilized the Chaos Mesh[10] to forcibly delete a pod and immediately restart a new one.
- **Abnormal pod scaling**. Kubernetes can automatically adjust the number of pods based on resource usage by the auto-scaling mechanism. However, due to misconfiguration or insufficient resources, pods may continuously adjust in the scaling-up and scaling-down states, which impairs the cluster's performance. We leveraged Chaos Mesh to simulate these activities.
- **Latency delay**. We utilized Istio's virtual service to inject a customized response latency into a specified microservice instance. For our experiments, the latency was set to 2,000 ms.
- **CPU hog**. Excessive CPU usage can significantly affect system performance, causing longer response times or rendering CPU-sensitive microservices unavailable. We simulated this scenario using Chaos Mesh, forcing a pod to reach 100% CPU utilization.
- **Memory leak**. Memory leaks occur when applications fail to release allocated memory properly. This issue results in a gradual increase in memory usage, eventually exhausting available memory and leading to performance degradation or system crashes. We also use the Chaos Mesh to inject memory leaks.

4.1.4. Baseline methods

MHP-RCA is a causal inference-based root cause localization method. To evaluate its efficacy, we compare it with state-of-the-art causal inference-based approaches, such as CausalRCA [23], MicroDiag [24], and MULAN [25]. Considering that MHP-RCA combines both metric and audit log data, we also compare it with two multimodal SOTA root cause localization methods: MULAN [25] and Nezha [26].

Additionally, to thoroughly assess the performance of MHP-RCA, we apply various causal structure learning methods (both linear and nonlinear) for comparison. For example, the PC and GES algorithms are effective in constructing linear causal relationships, while the CAM algorithm is suitable for both linear and nonlinear causal relationships. Similar to MHP-RCA, we integrate the causal graphs generated by these causal learning methods with personalized parameters in PPR, serving as causal weights for root cause identification. It is important to note that we also incorporate audit logs and anomalous events in this process. The baseline methods are listed as follows:

- **CausalRCA** [23]: It uses a gradient-based method for causal learning to generate weighted graphs. Root metrics are identified using the PageRank algorithm.
- **MicroDiag** [24]: It employs DirectLiNGAM to infer causality from non-Gaussian linear models. The graph is first weighted using the Pearson correlation coefficient between two metrics, followed by root metric ranking via the PageRank algorithm.
- **Nezha** [26]: Nezha is an interpretable RCA approach for microservices. It unifies multi-modal data into event representations for graph constructions and identifies root causes by comparing fault-free and fault-suffering patterns.
- **MULAN** [25]: MULAN extracts shared and modality-specific features across modalities through contrastive learning and constructs a causal graph using an attention mechanism. Based on the causal graph, it identifies root causes through random walk-based fault propagation analysis.
- **Granger Causality (GC)** [11]: It is a time series analysis method commonly used for causal graph construction. We use the χ^2 test for causality assessments.
- **Peter-Clark (PC)** [27]: It uses conditional independence testing to analyze the correlation between variables and subsequently determine the causal graph.
- **Greedy Equivalence Search (GES)** [28]: It is a score-based causal method where the Bayesian Information Criterion (BIC) is used as the scoring criterion.
- **Causal Additive Model (CAM)** [29]: It is a method based on the Structural Causal Model (SCM). It considers linear and nonlinear interactions among multiple factors, identifying causal relationships through the SCM that fit with Gaussian errors.
- **Linear Non-Gaussian Acyclic Model (LiNGAM)** [30]: It assumes non-Gaussian causal relationships and aims to find a model that represents the causal relationships between variables as linear equations. We implement the LiNGAM algorithm based on the Independent Component Analysis (ICA).

4.1.5. Evaluation metrics

To evaluate the effectiveness of different methods, we use the following evaluation metrics:

$AC@k$ (Top-k accuracy) indicates the probability that the first k elements in the final list correctly identify the true cause. A higher $AC@k$ indicates that the algorithm is better at locating the correct root cause. The evaluation metrics are set with $k = 1$, $k = 3$, $k = 5$, and $k = 10$. Assuming that $R[i]$ is an element in the final root cause list, V_{rc} is the set of real root causes, and $AC@k$ is defined as:

$$AC@k = \frac{1}{|A|} \sum_{a \in A} \frac{\sum_{i<k} f(R[i])}{\min\left(k, |V_{rc}|\right)}, \qquad f(R[i]) = \begin{cases} 1 & \text{if } R[i] \text{ in } V_{\text{rc}} \\ 0 & \text{otherwise} \end{cases}. \tag{20}$$

$Avg@k$ (Top-k average accuracy) denotes the average accuracy performance, which is defined as:

$$Avg@k = \frac{1}{k|A|} \sum_{a \in A} \sum_{1 \le k \le n} AC@k. \tag{21}$$

4.1.6. Statistical testing

To evaluate whether the performance improvement of our method over the competing method is statistically significant, we use the *t-test* to check the differences. We use $Avg@5$ as the performance score of each RCA method and construct the corresponding null hypothesis (no significance) and alternative hypothesis (significance). If the *p-value* of the *t-test* is less than 0.05 by default, then the null hypothesis is rejected, and the result is considered statistically significant.

4.2. Performance comparison (RQ1)

Table 3 presents the performance of different methods across four datasets. On dataset $\mathcal{A}$, MHP-RCA achieves an $AC@1$ of 38.57%, showing the best performance across all evaluation metrics. On datasets

[10] https://chaos-mesh.org

Table 3
Overall performances on four datasets.

Dataset	Method	$AC@1$	$AC@3$	$AC@5$	$AC@10$	$Avg@3$	$Avg@5$	$Avg@10$	P-value
$\mathcal{A}$	MicroDiag	0.2053	0.4421	0.7737	0.9211	0.3281	0.5105	0.7526	3.43e−03
	CausalRCA	0.1857	0.5286	0.7571	**1.0000**	0.3738	0.5129	0.7164	4.94e−02
	Nezha	0.2427	0.6176	0.7623	**1.0000**	0.4320	0.6558	0.7356	1.75e−02
	MULAN	0.3367	0.6605	0.8605	**1.0000**	0.4988	0.6448	0.7827	6.14e−02
	GC-based	0.0976	0.3171	0.6390	0.9024	0.3114	0.5878	0.7244	2.63e−03
	CAM-based	0.1316	0.4684	0.7000	0.9474	0.3544	0.5369	0.7711	1.15e−02
	PC-based	0.2071	0.5786	0.8500	**1.0000**	0.4024	0.5743	0.7793	4.13e−03
	LiNGAM-based	0.1755	0.5127	0.7737	0.9211	0.3556	0.5684	0.7058	8.47e−02
	GES-based	0.1929	0.5143	0.7571	**1.0000**	0.3667	0.5157	0.7150	9.58e−03
	MHP-RCA	**0.3857**	**0.8072**	**0.9214**	**1.0000**	**0.6143**	**0.7300**	**0.8550**	–
	Improvement	+4.9%	+19.0%	+6.1%	0%	+10.6%	+7.4%	+7.2%	–
$\mathcal{B}$	MicroDiag	0.0889	0.3244	0.4309	0.8746	0.2472	0.3142	0.5027	1.71e−02
	CausalRCA	0.1038	**0.4180**	0.5360	0.8298	0.2765	0.3933	0.5450	3.11e−02
	Nezha	0.1678	0.3044	0.5124	0.7911	0.2359	0.3569	0.5477	4.47e−02
	MULAN	0.2059	0.4060	0.5549	0.8559	0.2784	0.3609	0.5662	2.16e−02
	GC-based	0.0899	0.2203	0.4067	0.7126	0.1549	0.2810	0.4446	1.14e−02
	CAM-based	0.0679	0.2598	0.3724	0.6171	0.1940	0.1996	0.4110	9.41e−03
	PC-based	0.1357	0.2393	0.4679	0.8357	0.1940	0.2893	0.4957	7.93e−02
	LiNGAM-based	0.0679	0.3321	0.4714	0.8893	0.2143	0.3080	0.5243	5.14e−02
	GES-based	0.0679	0.2250	0.4107	0.7678	0.1452	0.2414	0.4382	1.13e−01
	MHP-RCA	**0.2143**	0.4036	**0.5893**	**0.9107**	**0.3155**	**0.4086**	**0.6040**	–
	Improvement	+0.9%	−1.6%	+3.4%	+1.1%	+3.70%	+1.5%	+3.8%	–
$\mathcal{C}$	MicroDiag	0.1053	0.3158	0.4561	0.8772	0.2164	0.2947	0.5211	4.95e−02
	CausalRCA	0.0952	0.3300	0.6614	0.8052	0.2797	0.3756	0.6084	1.58e−01
	Nezha	0.1960	0.3400	0.5840	0.7932	0.2628	0.4050	0.7049	6.50e−02
	MULAN	0.2012	0.3174	0.6389	0.8335	0.2506	**0.4820**	0.7580	1.39e−01
	GC-based	0.0952	0.2963	0.4259	0.8704	0.1975	0.2741	0.5037	7.97e−02
	CAM-based	0.0784	0.2745	0.3922	0.7627	0.1830	0.2510	0.4843	4.83e−02
	PC-based	0.1111	0.3548	0.5741	**0.8890**	0.2654	0.3741	0.6902	2.53e−01
	LiNGAM-based	0.0952	0.3095	0.4286	0.8810	0.2063	0.2809	0.5095	1.06e−01
	GES-based	0.0816	0.3530	0.3878	0.7571	0.1905	0.2571	0.4878	1.62e−01
	MHP-RCA	**0.2235**	**0.3904**	**0.6813**	0.8482	**0.2941**	0.4665	**0.7587**	–
	Improvement	+1.9%	+1.5%	+1.1%	−4.1%	+2.9%	−1.6%	+0.1%	–
$\mathcal{D}$	MicroDiag	0.1482	0.2894	0.6053	0.7889	0.1704	0.3816	0.5321	4.21e−02
	CausalRCA	0.2449	0.3694	0.5971	0.8328	0.3026	0.4212	0.6714	9.68e−02
	Nezha	0.2543	0.4068	0.6408	0.8015	0.3590	0.4272	0.6053	8.25e−02
	MULAN	0.2759	0.4212	0.6719	0.8275	0.3631	0.4428	0.6564	4.89e−01
	GC-based	0.1653	0.3080	0.4944	0.7779	0.2510	0.3631	0.5589	9.51e−03
	CAM-based	0.1976	0.3005	0.5983	0.8145	0.2263	0.3755	0.5867	2.56e−02
	PC-based	0.0822	0.2571	0.4887	0.6824	0.1477	0.3205	0.6034	3.71e−02
	LiNGAM-based	0.1792	0.2767	0.5181	0.7485	0.2209	0.3903	0.5609	2.08e−02
	GES-based	0.1039	0.2589	0.4619	0.7046	0.1840	0.3805	0.5432	8.50e−03
	MHP-RCA	**0.2979**	**0.4395**	**0.7140**	**0.8463**	**0.3850**	**0.4825**	**0.6970**	–
	Improvement	+2.2%	+1.8%	+4.2%	+1.3%	+2.2%	+4.0%	+2.6%	–

$\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$, all methods exhibited lower performance in Top-1 accuracy. This decline is attributed to the increased complexity inherent in localizing root causes within a system characterized by a greater number of services. While MHP-RCA may not consistently lead in every individual metric, it excels notably in $AC@1$, $AC@5$, $Avg@3$, and $Avg@10$, showcasing superior overall performance.

However, it is important to note that MHP-RCA shows suboptimal results in dataset $\mathcal{C}$. Because, unlike other benchmarks, the sock-shop architecture includes a RabbitMQ middleware microservice in its chain, which facilitates asynchronous message transmission between microservices. Consumers retrieve messages from RabbitMQ, potentially process the data (e.g., transformation or validation), and then store it in MongoDB. During this process, factors such as message delays, retry mechanisms, or variations in consumer processing speed may result in discrepancies between the recorded timestamps and the actual event sequence. Such disruptions may introduce bias in causal inference, ultimately affecting the reliability of RCA. This also highlights the limitation of MHP-RCA in effectively handling anomalies in asynchronous communication.

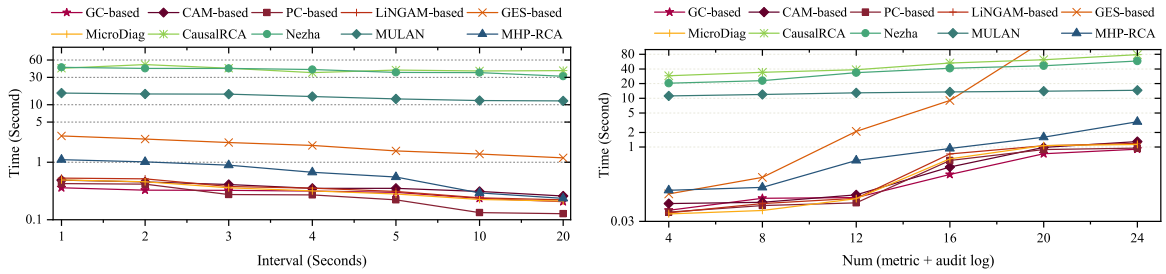
In summary, MHP-RCA exhibits the highest $AC@1$ performance across four datasets, achieving an average of 27.9%. Compared to the best-performing baseline approaches, it shows average improvements of 2.5% in $AC@1$, 3.7% in $AC@5$, and 2.8% in $Avg@5$, respectively. We also compared the statistical significance of MHP-RCA with other methods. The *p-value* in Table 3 indicates that MHP-RCA shows superior significance on datasets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{D}$.

4.3. Impact of different anomaly cases (RQ2)

Since various types of anomalies may occur in microservice-based systems, we evaluate the performance of root cause localization methods across different anomaly cases, such as continuous pod restarting, abnormal pod scaling, latency delay, CPU hog, and memory leak, as shown in Table 4.

In the case of continuous pod restarting, MHP-RCA outperforms baseline approaches on four datasets. Specifically, on dataset $\mathcal{A}$, MHP-RCA achieves a $AC@1$ of 32.14%, showing a significant improvement compared to other methods. This advantage is largely due to audit logs effectively capturing pod restart events at the process level, enabling accurate root cause identification.

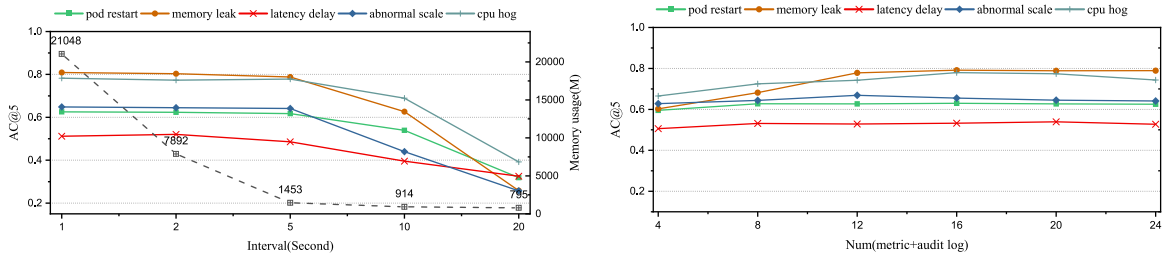
In the case of abnormal pod scaling, CausalRCA performs slightly better than MHP-RCA in terms of $AC@5$ on dataset $\mathcal{B}$, which is attributed to the longer interval between abnormal scaling. The causality generated by gradient learning can provide more precise localization. However, MHP-RCA demonstrates better overall performance in this case.



(a) Impact of time intervals over execution time

(b) Impact of numbers of data types over execution time

Fig. 5. Effect of different parameters on localization time.



(a) Impact of time intervals on localization performance.

(b) Impact of data types on localization performance.

Fig. 6. Effect of different parameters on localization performance.

In the case of latency delay, MHP-RCA performs slightly worse than Nezha. Because Nezha integrates three modalities of data, enabling better modeling of fault propagation relationships through the complete upstream and downstream information. Additionally, Nezha focuses the localization process on differences in abnormal patterns rather than conducting an exhaustive search for potential root causes by comparing failure-free and failure-suffer states.

In the case of memory leak and CPU hog, where anomalies are not clearly reflected in audit logs, the key data for identifying root causes are metrics. In the case of memory leaks, anomalies are clearly reflected in metrics, MHP-RCA can highlight anomalous features by extracting anomalous events, with more precise results. However, in the case of CPU hog, MHP-RCA's performance on datasets $\mathcal{B}$ and C is suboptimal because some microservices are not sensitive to CPU utilization, resulting in subtle fluctuations in metrics.

4.4. Efficiency comparison (RQ3)

The execution time is influenced by the interval for data collection and the number of data types. The interval represents the frequency of metrics acquisition and data types include various metrics (e.g., CPU, memory, and network usage) and audit logs (e.g., error exceptions and container startup events). Generally, a smaller interval results in finer granularity of data collection, but also leads to a larger volume of data, which in turn prolongs the execution time. From Fig. 5(a), we can see that most RCA methods can complete the localization within one second. Among these methods, the GES-based approach takes longer to generate causal graphs than others. Deep learning-based methods, such as CausalRCA, Nezha, and MULAN, are time-consuming during the training phase for constructing dependency graphs, surpassing other methods significantly. Although MHP-RCA may not achieve the highest efficiency, its time consumption remains acceptable when considering the trade-off between accuracy and efficiency.

The execution time is also influenced by the number of data types involved. As shown in Fig. 5(b), as data types increase, the complexity of constructing causal graphs rises, leading to longer durations for localization across all compared methods. CausalRCA exhibits the most significant increase, whereas the GES-based method shows the steepest rise in the rate of time increment. Notably, when handling 20 data types, it requires over 80 min to localize the root cause. In comparison, the increment rate and overall time consumption for MHP-RCA are more manageable. Even with 24 data types, MHP-RCA is capable of completing the localization within a few seconds. It maintains a practical balance between thoroughness and speed, making it a viable option for complex, data-rich environments.

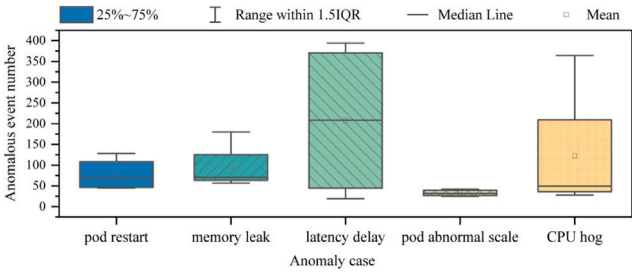


Fig. 7. Number of anomalous events generated by different anomaly cases.

4.5. Parameter analysis (RQ4)

One crucial factor of the localization is the data length, which impacts both efficiency and effectiveness. Insufficient data length may fail to capture abnormal patterns, leading to poor results comprehensively. As shown in Fig. 6(a), $AC@5$ remains stable or slightly decreases after reaching an interval of five seconds. At this point, the number of generated anomalous events varies for different abnormal cases. As depicted in Fig. 7, the largest number of anomalous events is generated in the case of latency delay. This is because the delay will affect the response time of both upstream and downstream microservices. In contrast, the pod scaling anomaly generates fewer anomalous events due to the extended intervals between scaling operations, necessitated by the process of terminating and starting containers.

In general, utilizing key data ensures effective localization, but if the input data is noisy, the final localization results will be poor. We select several key data types for each microservice and gradually add other commonly used data. As shown in Fig. 6(b), when the number of

Table 4
Performances of different anomaly cases.

Anomaly case	Method	Dataset $\mathcal{A}$			Dataset $\mathcal{B}$			Dataset $\mathcal{C}$			Dataset $\mathcal{D}$		
		$AC@1$	$AC@3$	$AC@5$	$AC@1$	$AC@3$	$AC@5$	$AC@1$	$AC@3$	$AC@5$	$AC@1$	$AC@3$	$AC@5$
Continuous pod restarting	MicroDiag	0.1504	0.2951	0.6271	0.1462	0.2105	0.5156	0.1453	0.2097	0.5129	0.1810	0.2607	0.4721
	CausalRCA	0.1429	0.5000	0.6429	0.1210	0.3939	0.5859	0.2340	**0.3663**	0.5309	0.2800	0.3678	0.5754
	Nezha	0.1524	0.5438	0.7445	0.2257	0.3931	0.4918	0.2383	0.2916	0.4055	0.2618	0.4042	0.5634
	MULAN	0.2665	0.4833	0.8186	0.1988	**0.4578**	0.5839	0.2318	0.2801	0.5735	0.2915	0.4325	0.6256
	GC-based	0.1289	0.3054	0.4639	0.0738	0.1925	0.2097	0.0738	0.1925	0.2097	0.1598	0.3474	0.4733
	CAM-based	0.1574	0.3726	0.5513	0.0348	0.1589	0.4184	0.0348	0.1589	0.4184	0.2575	0.3281	0.5820
	PC-based	0.1071	0.2500	0.4643	0.1453	0.2097	0.5129	0.1462	0.2105	0.5156	0.0986	0.3023	0.4285
	LiNGAM-based	0.2500	0.6429	**0.8571**	0.2045	0.3184	0.4689	0.2045	0.3184	0.4689	0.2101	0.2587	0.4602
	GES-based	0.1786	0.3214	0.3571	0.1812	0.3156	0.4915	0.1812	0.3156	0.4915	0.1682	0.2538	0.3459
	MHP-RCA	**0.3214**	**0.7143**	**0.8571**	**0.2579**	0.3427	**0.6221**	**0.2579**	0.3427	**0.6221**	**0.3310**	**0.4742**	**0.6786**
	Improvement	+5.5%	+7.1%	0%	+5.3%	−11.5%	+3.6%	+1.9%	−2.4%	+4.9%	+3.0%	+4.2%	+5.3%
Abnormal pod scaling	MicroDiag	0.0778	0.4032	0.9396	0.1277	0.1447	0.3370	0.0000	0.3612	0.6458	0.1131	0.1627	0.5394
	CausalRCA	0.0000	0.4643	0.9882	0.1071	0.2679	**0.4643**	0.0000	0.4332	0.7581	0.1020	0.1958	0.5474
	Nezha	0.1598	0.6481	**1.0000**	0.1406	0.2070	0.3123	0.1198	0.3611	0.7163	0.0769	0.1869	0.5160
	MULAN	0.1948	0.7270	**1.0000**	0.1781	0.2672	0.3392	0.1906	0.2665	0.7433	0.1578	0.2257	0.5678
	GC-based	0.1093	0.5628	0.7541	0.0849	0.1149	0.2158	0.0000	0.2221	0.4226	0.0774	0.1478	0.2705
	CAM-based	0.1286	0.6491	0.8494	0.0849	0.1796	0.3135	0.0745	0.2989	0.4296	0.1371	0.1756	0.4738
	PC-based	0.0357	0.4286	0.9247	0.1623	0.2007	0.4337	0.1597	0.2974	0.7802	0.0466	0.0942	0.3687
	LiNGAM-based	0.1071	0.6786	0.9741	0.0000	0.1964	0.2500	0.0827	0.2791	0.6701	0.0370	0.1513	0.4319
	GES-based	0.1786	0.6429	0.8535	0.0179	0.0893	0.1786	0.1597	0.4085	0.7409	0.0574	0.1016	0.3390
	MHP-RCA	**0.2143**	**0.8929**	0.9917	**0.1786**	**0.2857**	0.4286	**0.1912**	**0.4722**	**0.8276**	**0.1742**	**0.2494**	**0.6453**
	Improvement	+2.7%	+16.5%	−0.8%	+0.1%	+1.8%	−3.6%	+0.1%	+3.9%	+4.7%	+2.7%	+2.4%	+7.8%
Latency delay	MicroDiag	0.0000	0.1119	0.2985	0.0000	0.2057	0.5225	**0.1376**	0.2492	**0.4753**	0.0445	0.2598	0.5591
	CausalRCA	0.0714	**0.4643**	**0.5000**	0.2312	0.4672	0.5787	0.1359	0.1997	0.4388	0.2300	0.2876	0.5097
	Nezha	0.1544	0.4112	0.3947	0.2436	0.3934	0.5886	0.0959	0.2901	0.4483	**0.2592**	**0.3875**	**0.6632**
	MULAN	0.2228	0.3908	0.4837	**0.3571**	0.5238	0.6023	0.0843	0.2676	0.4213	0.2300	0.2876	0.5597
	GC-based	0.1849	0.4197	0.4497	0.0857	0.1397	0.4060	0.0000	0.2745	0.3347	0.1607	0.2266	0.3596
	CAM-based	0.0619	0.2514	0.3942	0.1250	0.2644	0.3333	0.0000	0.1954	0.2754	0.1700	0.2356	0.5809
	PC-based	0.0000	0.2143	0.4286	**0.3571**	0.3750	0.3929	0.1099	**0.3132**	0.4129	0.0000	0.2095	0.5630
	LiNGAM-based	0.1071	0.1071	0.3571	0.1250	**0.5357**	0.5893	0.1068	0.1912	0.4497	0.1446	0.2217	0.5195
	GES-based	0.1071	0.1429	0.3571	0.1653	0.2500	0.3571	0.0956	0.1903	0.2995	0.1069	0.2566	0.4902
	MHP-RCA	**0.2500**	0.3929	**0.5000**	**0.3571**	0.5000	**0.6607**	0.0923	0.3115	0.4661	0.2463	0.3631	0.5907
	Improvement	+2.7%	−7.1%	0%	0%	−3.6%	+5.8%	−5.3%	−4.6%	−5.4%	−2.9%	−10.0%	−3.1%
Memory leak	MicroDiag	0.1913	0.4445	0.8116	0.1208	0.1227	0.4046	0.1641	0.4135	0.6902	0.2739	0.3573	0.6998
	CausalRCA	0.3571	0.5000	0.8214	0.1692	0.3398	0.5896	0.2584	0.4656	0.6786	0.3011	0.4255	0.6587
	Nezha	0.4023	0.7035	0.8246	0.1761	0.2695	0.6136	0.3184	0.3288	0.7739	0.3406	0.4468	0.6501
	MULAN	0.5703	0.8514	**1.0000**	0.1930	**0.4484**	0.6150	0.3646	0.3927	0.7841	0.3294	0.4393	0.7497
	GC-based	0.1847	0.4782	0.7993	0.1481	0.3131	0.5279	0.1418	0.2165	0.4901	0.2101	0.3074	0.5788
	CAM-based	0.1928	0.7857	0.9254	0.0701	0.2420	0.5545	0.1047	0.3814	0.4782	0.2101	0.3074	0.5788
	PC-based	0.1786	0.8361	**1.0000**	0.0179	0.0536	0.3929	0.2328	0.3291	0.6834	0.2196	0.3223	0.5788
	LiNGAM-based	0.2500	0.5357	0.7500	0.1786	0.2857	0.3929	0.1376	0.3852	0.5892	0.2705	0.3498	0.5788
	GES-based	0.2500	0.6429	0.7857	0.1607	0.2143	0.4821	0.0971	0.4634	0.5412	0.1232	0.2888	0.5527
	MHP-RCA	**0.5714**	**0.9286**	**1.0000**	**0.2321**	0.4286	**0.6250**	**0.3956**	**0.4958**	**0.7935**	**0.3687**	**0.4663**	**0.7738**
	Improvement	+1.0%	+7.5%	0%	+2.9%	−2.0%	+3.5%	+3.1%	+3.0%	+1.0%	+2.8%	+2.0%	+2.4%
CPU hog	MicroDiag	0.2911	0.5683	**1.0000**	0.0000	0.3875	0.5454	0.1939	0.3432	0.6483	0.1286	0.4064	0.7559
	CausalRCA	0.3214	0.7500	0.8214	0.0207	0.2974	0.5444	0.1497	0.3297	0.6661	0.3112	0.5704	0.6944
	Nezha	0.3444	0.7812	0.8476	0.0530	0.2590	0.5558	**0.2076**	0.4283	0.5760	0.3329	0.6084	0.8111
	MULAN	0.4290	0.8498	**1.0000**	0.1025	0.3329	0.6343	0.1349	0.4401	0.6725	0.3547	**0.6456**	0.8259
	GC-based	0.2789	0.6279	0.8163	0.0000	0.2527	0.4902	0.1347	0.3152	0.3578	0.2318	0.5576	0.8092
	CAM-based	0.3765	0.6723	**1.0000**	0.1066	0.1501	0.4625	0.1923	0.4751	0.5487	0.2133	0.4557	0.7759
	PC-based	0.2500	0.7857	0.9286	0.0893	0.3571	0.5893	0.0146	**0.4887**	0.5632	0.0462	0.3571	0.5046
	LiNGAM-based	0.3214	0.6786	0.8571	0.0000	**0.3929**	**0.6429**	0.1453	0.3442	0.5684	0.2336	0.4021	0.6000
	GES-based	0.2500	0.5714	0.8571	0.0893	0.3036	0.5454	0.0135	0.3871	0.4987	0.0639	0.3935	0.5815
	MHP-RCA	**0.4714**	**0.8571**	**1.0000**	**0.1250**	0.3393	0.5893	0.1806	0.4416	**0.6974**	**0.3636**	0.6239	**0.8516**
	Improvement	+4.2%	+1.8%	0%	+1.8%	−5.4%	−5.4%	−2.0%	−4.2%	+2.5%	+0.9%	−2.2%	+2.5%

data types reaches 16, it has a subtle performance impact on anomaly cases like pod restarting, latency delay, and abnormal scaling. This is because MHP-RCA effectively filters out noise from uncommon data by extracting anomalous events during the processing of metrics and audit logs. In cases of Memory leak and CPU hog, the performance was not optimal with fewer data types. This is because microservice systems involve resource sharing, requiring more data to capture causal relationships accurately.

Additionally, the generation of anomalous events is influenced by the Birch clustering threshold. Setting this threshold too low results in an overabundance of anomalous events and potentially irrelevant data. Conversely, an overly high threshold can lead to an underrepresentation of anomalous events, resulting in a limited causal graph that may overlook the actual root cause. Due to differences in metric levels, we do not standardize the unit of service-level metrics (θ_1) and system-level metrics (θ_2). Instead, we set different Birch clustering thresholds. The final result is shown in Fig. 8, indicating that θ_1=100 and θ_2=1000 yield the highest localization performance, allowing for the effective generation of anomalous events for both service-level and system-level metrics across various anomaly scenarios.

Table 5
Ablation studies.

Anomaly cases	Ablation	Dataset $\mathcal{A}$			Dataset $\mathcal{B}$			Dataset $\mathcal{C}$			Dataset $\mathcal{D}$		
		$AC@1$	$AC@3$	$AC@5$	$AC@1$	$AC@3$	$AC@5$	$AC@1$	$AC@3$	$AC@5$	$AC@1$	$AC@3$	$AC@5$
Continuous pod restarting	w/o Birch	0.2998	0.6865	0.8304	0.2295	0.4595	**0.5953**	0.2352	0.3088	0.6092	0.2905	0.4436	0.6602
	w/o PPR	0.3173	0.6588	**0.8478**	0.2245	0.4512	0.5858	**0.2500**	0.3188	0.6060	0.2628	0.4406	0.6501
	w/o Hawkes	0.2639	0.6388	0.7581	0.1940	0.4228	0.5449	0.2257	0.2893	0.5923	0.2465	0.4215	0.6214
	w/o Alignment	**0.3207**	**0.6975**	0.8399	**0.2435**	**0.4674**	0.5827	0.2425	**0.3347**	**0.6160**	**0.2946**	**0.4636**	**0.6721**
Abnormal pod scaling	w/o Birch	0.1947	0.8485	0.9150	**0.0978**	0.2627	0.4617	0.1801	0.4673	**0.7917**	0.1232	0.2197	0.6112
	w/o PPR	**0.2004**	0.8409	**0.9694**	0.0803	0.2287	**0.4616**	0.1855	**0.4690**	0.7759	0.0708	0.2129	**0.6258**
	w/o Hawkes	0.1544	0.8042	0.8671	0.0705	0.2248	0.4595	0.1695	0.3823	0.7613	0.1305	0.2097	0.6125
	w/o Alignment	0.1703	**0.8816**	0.9602	0.0772	**0.2662**	0.4573	**0.1907**	0.4697	0.4690	**0.1657**	**0.2250**	0.6227
Latency delay	w/o Birch	0.2352	**0.3865**	**0.4982**	0.3260	0.4552	0.6486	0.0523	0.1851	**0.4163**	0.2385	0.3574	0.5921
	w/o PPR	0.2386	0.3744	0.4785	0.3165	0.4748	0.6324	**0.0883**	0.1596	0.3897	0.2275	0.3699	0.5936
	w/o Hawkes	0.2072	0.3711	0.4612	0.2974	0.4531	0.6056	0.0314	0.1535	0.3704	0.2352	0.3699	0.6038
	w/o Alignment	**0.2431**	0.3824	0.4979	**0.3559**	**0.4903**	**0.6594**	0.0592	**0.1991**	0.4137	**0.2451**	**0.3709**	**0.6096**
Memory leak	w/o Birch	0.5574	0.9006	0.9530	**0.2290**	0.4027	0.5864	**0.3934**	0.4527	**0.7781**	0.3483	0.4519	0.7605
	w/o PPR	0.5367	0.9124	0.9507	0.2107	0.4040	0.6095	0.3764	0.4819	0.7760	0.3238	0.4226	0.7317
	w/o Hawkes	0.5351	0.9283	0.9051	0.1768	0.3792	0.5739	0.3507	0.3977	0.7211	0.3572	0.4424	0.7072
	w/o Alignment	**0.5704**	**0.9414**	**1.0000**	0.2213	**0.4164**	**0.6167**	0.3856	**0.4956**	0.7741	**0.3662**	**0.4571**	**0.7674**
CPU hog	w/o Birch	0.4635	0.8306	0.9876	0.1014	**0.3343**	**0.5818**	0.1713	0.2827	0.6526	0.3234	0.6026	0.8418
	w/o PPR	0.4649	**0.8337**	0.9711	0.1116	0.3234	0.5711	**0.1747**	0.2894	0.6260	0.3353	0.5768	0.8036
	w/o Hawkes	0.4194	0.8028	0.9273	0.0569	0.2710	0.5398	0.1454	0.2765	0.6184	**0.3483**	0.6026	0.8071
	w/o Alignment	**0.4702**	0.8335	**1.0000**	**0.1167**	0.3159	0.5568	0.1714	**0.2996**	**0.6885**	0.3165	**0.6239**	**0.8444**

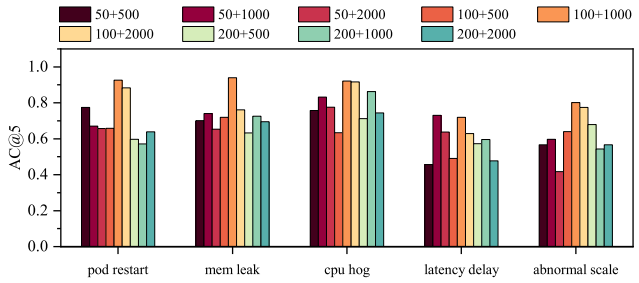


Fig. 8. Impact of Birch clustering parameters on localization performance.

4.6. Ablation study (RQ5)

To validate the effectiveness of individual components in the MHP-RCA framework, we conducted a series of ablation experiments. These experiments systematically replaced or removed certain modules to evaluate their contributions to root cause analysis performance. The following modifications were applied: (*i*) we replaced the Birch clustering algorithm with a basic 3-sigma anomaly detection method to examine the necessity of an advanced anomaly detection; (*ii*) we removed the Hawkes process and relied solely on the Granger causality to assess the importance of temporal modeling in capturing causal relationships; (*iii*) we substituted the Personalized PageRank algorithm with the standard PageRank algorithm to explore the benefits of personalized propagation mechanisms, and (*iv*) we compared the performance with and without alignment between metrics and logs to investigate its impact on diagnosis accuracy. The results, as shown in Table 5, indicate that each component enhances localization accuracy, with the Hawkes process contributing the most and multimodal alignment playing a relatively smaller but still beneficial role.

In addition, audit logs provide detailed information at the container process level, while system logs contain default output logs from Kubernetes, such as host-level information. We evaluate the significance of audit logs in the localization process and compare the impact of using audit logs, system logs, and scenarios without any logs on localization performance. From Fig. 9, it can be observed that audit logs significantly improve localization performance in cases such as continuous pod restarting and abnormal pod scaling. However, the benefits of using audit logs are not uniform across all types of anomalies. For example, in CPU hog scenarios, we observe that localization performance is actually better without any logs. This can be explained by the fact that audit logs tend to record abnormal behaviors of pods resulting from resource limitations, which may introduce interference and thereby negatively impact the accuracy of root cause localization.

4.7. Robustness analysis (RQ6)

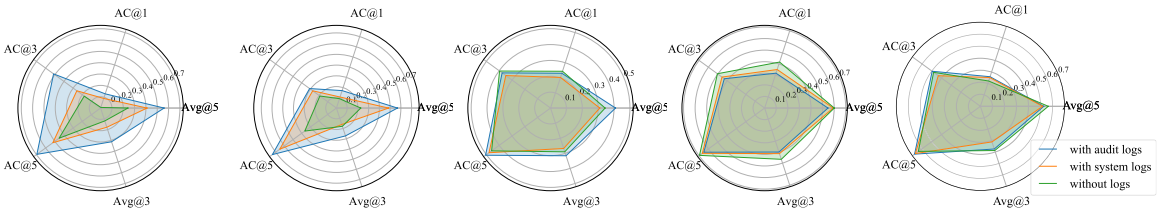
During data collection, issues such as network interruptions or service restarts may lead to data incompleteness, which can negatively impact the effectiveness of root cause analysis. Therefore, it is crucial for root cause analysis models to exhibit robustness against noise. We apply masking to multimodal monitoring data to simulate missing data. Specifically, a proportion of metrics and logs is randomly selected and altered. For metrics, the original data is replaced with zeros, while for audit logs, entries in the original log sequence are randomly deleted.

Fig. 10 illustrates the experimental results of MHP-RCA on different datasets under varying data missing rates. As the missing rate increases, the localization accuracy declines across all cases. When the missing rate remains below 10%, the performance loss is minimal, with the AC@5 metric decreasing by 0.9%, 1.2%, 4.6% and 6.2% on the four datasets, respectively. However, when the missing rate exceeds 20%, the accuracy of the analysis is significantly affected. This is because incomplete data alters the distribution of point events, and the causal graph generated from these events fails to accurately guide the fault propagation process, leading to degraded analysis performance. Despite the decline in performance caused by data incompleteness, MHP-RCA still outperforms GC-based methods, CAM-based methods, and MicroDiag in scenarios with a missing rate of 10%. These results validate the robustness and effectiveness of the proposed method for root cause analysis, even under conditions of data incompleteness.

4.8. Threats to validity

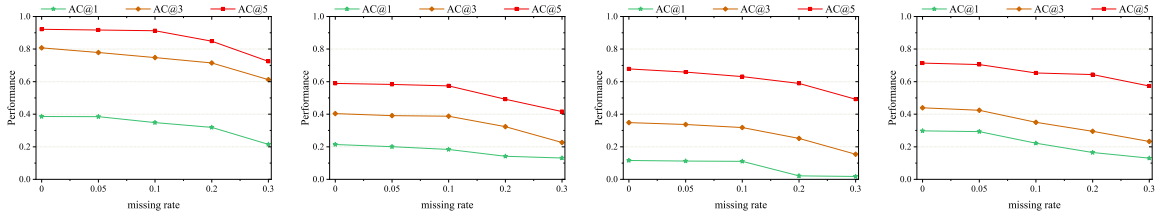
We have identified three categories of threats to the root cause localization method proposed in this paper.

The internal validity threat pertains to the potential impact of confounding variables, which typically arises from implementation. To ensure reliability and consistency, we minimized this threat by conducting repeated experiments to confirm that results were primarily influenced by the manipulated variables and not by external factors.



(a) Continuous pod restarting (b) Abnormal pod scaling (c) Latency delay (d) Memory leak (e) CPU hog

Fig. 9. Localization performances with/without audit logs and system logs.



(a) Performance on Dataset $\mathcal{A}$ (b) Performance on Dataset $\mathcal{B}$ (c) Performance on Dataset $\mathcal{C}$ (d) Performance on Dataset $\mathcal{D}$

Fig. 10. Experiments on incomplete datasets.

The threat to external validity pertains to the generalizability of the result beyond the experimental setup. Building complex infrastructure and conducting repeated experiments across multiple testbeds is highly costly, making it impractical for our research. We mitigated this threat by using diverse datasets, including different types of faults, and implementing widely recognized microservice architectures. The four benchmark systems used in the experiments are widely used in DevOps. Furthermore, because various anomalies can potentially impact the localization result, we injected different cases of anomalies to evaluate the effectiveness of different methods. Experiments on different benchmarks and various anomaly cases can help improve the applicability of our method to a wide range of real-world scenarios.

The construct validity threat to the effectiveness mainly lies in the selection of hyperparameters and evaluation metrics. In Section 4, we analyzed the hyperparameters in MHP-RCA, selecting appropriate parameters to improve localization performance. Additionally, we employed commonly used evaluation metrics in RCA problems.

5. Discussion

Currently, root cause localization methods are gradually transitioning from a single modality to multiple modalities. MHP-RCA, by combining various types of data, can not only detect latency faults but also identify other types of anomalies such as process anomalies and configuration errors, making it better suited for complex microservice architectures. Compared to traditional correlation analysis, MHP-RCA can extract hidden relationships from metrics and audit log data, helping engineers understand the complex interactions within the system, and thereby accurately pinpointing the root cause of faults. Additionally, it is worth mentioning that the root cause localization methods proposed by MHP-RCA can be applied not only in the field of microservices but also in performance analysis of large-scale distributed systems by collecting appropriate data monitoring (such as metrics and logs collected by ELK tools), where related causal inference and graph analysis methods can be utilized.

Limitations: Limitations: Despite its advantages, MHP-RCA has several limitations. (*i*) MHP-RCA relies on high-quality data. The completeness and accuracy of the data directly affect the reliability of the localization results. (*ii*) While MHP-RCA has optimized for time cost and computational resources, its efficiency may still degrade as the system scale increases. Further advancements are necessary to enhance its processing efficiency for larger systems. (*iii*) Although MHP-RCA demonstrated good performance in different scenarios, its dataset generalizability remains a limitation. The experimental evaluation primarily focuses on resource-related issues, process-related issues, and network-related issues, which, while common, do not fully capture the complexities of real-world microservices environments.

6. Related works

6.1. Log-based methods

Distributed applications deployed in communication networks usually generate a large volume of log messages during runtime. These logs document the state of servers and various operations carried out within microservice systems. With the advancement of data analysis techniques, various log-based root cause localization methods have been proposed. Cinque et al. [31] developed a performance monitoring tool specifically designed to target trace logs generated by 'request–response' messages. This tool catalogs details across various fault categories, facilitating comprehensive fault analysis. Xu et al. [32, 33] analyzed event logs by parsing log patterns. Then they established anomaly detection and fault recognition models from historical data. Aggarwal et al. [34] modeled the error rate of microservices as a multivariate sequence of events based on logs. They utilized the Granger causality test to construct causal relationships between services, followed by the PageRank to rank potentially faulty components, thereby aiding in fault localization. SwissLog [35] initially constructs a graph to represent the logging relationships across distributed components. It further enhances this analysis by applying temporal and semantic embeddings to the logging sequences, culminating in the RCA through a heuristic search algorithm.

Most log analysis methods focus only on static correlations between events, overlooking the temporal dynamics within event sequences, which prevents them from fully capturing the propagation paths or root causes of anomalies.

6.2. Metric-based methods

In recent years, some researchers have noticed that the dependencies (invocations or causal relationships) between microservices can be represented as graphs. In this context, numerous metrics-based

approaches have been proposed, which can be further categorized into two types: topology-based and causality-based methods.

Topology-based methods. These methods use the topological data of microservice invocations. MicroRCA [8] builds a service dependency graph and calculates the correlation between metrics using the Pearson coefficient. It then identifies root causes with the PageRank algorithm [21]. However, faults in multi-tiered systems occur not only at the service level but also at the physical layer. MS-Rank [7] addresses this by dynamically updating metric weights to handle different anomalies. Machine learning methods are also widely used. Brandon et al. [36] created fault patterns from predefined fault injections and classified faults by comparing abnormal patterns. MicroHECL [6] employs machine learning techniques to analyze metrics from invocation graphs, enabling the detection of service anomalies. Topology graphs are useful for representing connections but are limited in capturing complex causal relationships, especially in scenarios involving multiple factors and interactions in system behavior. As a result, many causality-based methods have been proposed.

Causality-based methods. Lin et al. [37] proposed Microscope, a dynamic root cause localization method, which adds abnormal services to candidate groups with the 3-Sigma algorithm. These services are then ranked based on metric correlation coefficients. MicroDiag [24] uses SCM and the Granger causality test to infer abnormal propagation directions and creates a causal dependency graph. A random walk algorithm is then applied to find the root cause. Sage [38] constructs a Causal Bayesian Network (CBN) to analyze latency and system metrics, using counterfactual inferences to locate the root cause. CausalRCA [23] uses a gradient-based method for causal structure learning and produces a weighted causal graph, with root causes identified through inference estimations.

Traditional causal detection methods (such as Granger causality) struggle to distinguish true causal relationships accurately. Moreover, relying solely on metric data may overlook other key information about system behavior (such as discrete events), leading to incomplete diagnostics.

6.3. Multi-modal-based methods

In recent years, there has been growing attention towards multi-modal-based methods. MULAN [25] relies on multimodal data and proposes a contrastive learning-based method for root cause localization. Through a KPI-aware attention mechanism, it evaluates modality reliability and generates a causal graph of fault propagation. Based on the graph, a random walk is used to identify root causes. DiagFusion [39] employs data augmentation to embed the multimodal data of service instances. It integrates deployment information and trace data to construct a dependency graph. A graph neural network is then used to localize the root cause. CloudRCA [40] utilizes multimodal data, including Key Performance Indicators (KPIs), logs, and topology, as input. It extracts features through log analysis and anomaly detection, which are then processed by a Knowledge-informed Hierarchical Bayesian Network (KHBN) model to infer root causes. Eadro [41] leverages multimodal data to model intra-service behaviors and inter-service dependencies. It uses multi-task learning to share knowledge, enabling an integrated approach for anomaly detection and root cause localization. Nezha [26] transforms metrics and log data into a unified event representation and extracts event patterns by constructing event graphs. It localizes root causes interpretably by comparing event patterns from the fault-free phase with those from the fault-suffering phase.

While multi-modal joint modeling significantly enhances feature representation, it also increases the dimensionality of the feature space, leading to higher complexity in training and inference. This becomes particularly challenging for large-scale systems, where computational resource consumption may become a bottleneck. In contrast, MHP-RCA does not rely on training data, enabling higher localization efficiency. Moreover, existing approaches often overlook failures at the container process level. MHP-RCA addresses this gap by integrating audit logs and metric data, achieving root cause localization at a finer granularity.

7. Conclusion

In this paper, we propose MHP-RCA, a root cause localization approach based on the multivariate Hawkes process. MHP-RCA begins by characterizing system anomalies as anomalous events, based on the analysis of metrics and audit logs. Next, causal graphs are constructed using the multivariate Hawkes process to infer the causal structure of fault propagation. Finally, the personalized PageRank is applied to conduct real-time root cause localization through random walks on the causal graph. Extensive experiments on microservice benchmarks demonstrate the effectiveness and practicality of our proposed method.

In the future, we plan to improve our approach from the following aspects. Firstly, feature engineering could be integrated into the data collection phase to reduce input dimensionality, thereby facilitating more efficient localization. Secondly, MHP-RCA only considers the numerical attributes of audit logs. We plan to integrate semantic attributes of audit logs into our framework. Furthermore, since MHP-RCA cannot cover all anomalies in microservice, more cases like anomalies in asynchronous calls will be considered in subsequent improvements.

CRediT authorship contribution statement

Zekun Zhang: Writing – review & editing, Writing – original draft. **Jian Wang:** Writing – review & editing. **Bing Li:** Writing – review & editing. **Yu Liu:** Writing – review & editing. **Hongyue Wu:** Writing – review & editing. **Patrick C.K. Hung:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (No. 2022YFF0902701), the National Natural Science Foundation of China (No. 62032016), and the Shenzhen Science and Technology Program under Grant CJGJZD20230724091659002.

Data availability

Data will be made available on request.

References

[1] Leilei Wang, Xiaoheng Deng, Jinsong Gui, Xuechen Chen, Shaohua Wan, Microservice-oriented service placement for mobile edge computing in sustainable internet of vehicles, IEEE Trans. Intell. Transp. Syst. 24 (9) (2023) 10012–10026.

[2] Peng Chen, Hongyun Liu, Ruyue Xin, Thierry Carval, Jiale Zhao, Yunni Xia, Zhiming Zhao, Effectively detecting operational anomalies in large-scale IoT data infrastructures by using a GAN-based predictive model, Comput. J. 65 (11) (2022) 2909–2925.

[3] Yujia Song, Ruyue Xin, Peng Chen, Rui Zhang, Juan Chen, Zhiming Zhao, Identifying performance anomalies in fluctuating cloud environments: A robust correlative-GNN-based explainable approach, Future Gener. Comput. Syst. 145 (2023) 77–86.

[4] Shijun Shen, Jiuling Zhang, Daochao Huang, Jun Xiao, Evolving from traditional systems to AIOps: design, implementation and measurements, in: 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications, AEECA, IEEE, 2020, pp. 276–280.

[5] Shenglin Zhang, Sibo Xia, Wenzhao Fan, Binpeng Shi, Xiao Xiong, Zhenyu Zhong, Minghua Ma, Yongqian Sun, Dan Pei, Failure diagnosis in microservice systems: A comprehensive survey and analysis, 2024, arXiv, arXiv:2407.01710.

[6] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, Zheshun 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, IEEE, 2021, pp. 338–347.

[7] Meng Ma, Weilan Lin, Disheng Pan, Ping Wang, Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications, in: 2019 IEEE International Conference on Web Services, ICWS, IEEE, 2019, pp. 60–67.
[8] Li Wu, Johan Tordsson, Erik Elmroth, Odej Kao, Microrca: Root cause localization of performance issues in microservices, in: NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2020, pp. 1–9.
[9] Thomas Josef Liniger, Multivariate Hawkes Processes (Ph.D. thesis), ETH Zurich, 2009.
[10] Mehrdad Farajtabar, Nan Du, Manuel Gomez-Rodriguez, Isabel Valera, Hongyuan Zha, Le Song, Shaping social activity by incentivizing users, in: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS '14, MIT Press, Cambridge, MA, USA, 2014, pp. 2474–2482.
[11] Clive W.J. Granger, Investigating causal relations by econometric models and cross-spectral methods, Econ.: J. Econ. Soc. (1969) 424–438.
[12] Li Wu, Jasmin Bogatinovski, Sasho Nedelkoski, Johan Tordsson, Odej Kao, Performance diagnosis in cloud microservices using deep learning, in: International Conference on Service-Oriented Computing, Springer International Publishing, Cham, 2020, pp. 85–96.
[13] Bing Tang, Feiyan Guo, Buqing Cao, Mingdong Tang, Kuanching Li, Cost-aware deployment of microservices for IoT applications in mobile edge computing environment, IEEE Trans. Netw. Serv. Manag. 20 (3) (2023) 3119–3134.
[14] Xiaodi Hou, Liqing Zhang, Saliency detection: A spectral residual approach, in: 2007 IEEE Conference on Computer Vision and Pattern Recognition, 2007, pp. 1–8.
[15] Tian Zhang, Raghu Ramakrishnan, Miron Livny, BIRCH: an efficient data clustering method for very large databases, SIGMOD Rec. 25 (2) (1996) 103–114.
[16] Min Du, Feifei Li, Guineng Zheng, Vivek Srikumar, Deeplog: Anomaly detection and diagnosis from system logs through deep learning, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1285–1298.
[17] Daryl J. Daley, David Vere-Jones, An introduction to the theory of point processes, second ed., in: Probability and Its Applications, Springer, New York, NY, 2003, p. 21, 471.
[18] Ruichu Cai, Siyu Wu, Jie Qiao, Zhifeng Hao, Keli Zhang, Xi Zhang, THPs: Topological hawkes processes for learning causal structure on event sequences, IEEE Trans. Neural Networks Learn. Syst. 35 (1) (2024) 479–493.
[19] Arthur P. Dempster, Nan M. Laird, Donald B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, J. R. Stat. Soc. Ser. B Stat. Methodol. 39 (1) (1977) 1–22.
[20] Ke Zhou, Hongyuan Zha, Le Song, Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes, in: Artificial Intelligence and Statistics, PMLR, 2013, pp. 641–649.
[21] Sergey Brin, Lawrence Page, The anatomy of a large-scale hypertextual web search engine, Comput. Netw. ISDN Syst. 30 (1–7) (1998) 107–117.
[22] Glen Jeh, Jennifer Widom, Scaling personalized web search, in: Proceedings of the 12th International Conference on World Wide Web, 2003, pp. 271–279.
[23] Ruyue Xin, Peng Chen, Zhiming Zhao, CausalRCA: Causal inference based precise fine-grained root cause localization for microservice applications, J. Syst. Softw. 203 (2023) 111724.
[24] Li Wu, Johan Tordsson, Jasmin Bogatinovski, Erik Elmroth, Odej Kao, Microdiag: Fine-grained performance diagnosis for microservice systems, in: 2021 IEEE/ACM International Workshop on Cloud Intelligence, CloudIntelligence, IEEE, 2021, pp. 31–36.
[25] Lecheng Zheng, Zhengzhang Chen, Jingrui He, Haifeng Chen, MULAN: Multi-modal causal structure learning and root cause analysis for microservice systems, in: Proceedings of the ACM Web Conference 2024, WWW '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 4107–4116.
[26] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, Zibin Zheng, 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, in: ESEC/FSE, New York, NY, USA, 2023, pp. 553–565.
[27] Peter Spirtes, Clark Glymour, An algorithm for fast recovery of sparse causal graphs, Soc. Sci. Comput. Rev. 9 (1) (1991) 62–72.
[28] David Chickering, Optimal structure identification with greedy search, J. Mach. Learn. Res. 3 (2002) 507–554.
[29] Peter Bühlmann, Jonas Peters, Jan Ernest, CAM: Causal additive models, high-dimensional order search and penalized regression, Ann. Statist. 42 (2013).
[30] Shohei Shimizu, Patrik O. Hoyer, Aapo Hyvärinen, Antti Kerminen, A linear non-Gaussian acyclic model for causal discovery, J. Mach. Learn. Res. 7 (72) (2006) 2003–2030.
[31] Marcello Cinque, Raffaele Della Corte, Antonio Pecchia, Microservices monitoring with event logs and black box execution tracing, in: 2021 IEEE World Congress on Services, SERVICES, 2021, pp. 12–12.
[32] Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael I Jordan, Detecting large-scale system problems by mining console logs, in: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, 2009, pp. 117–132.
[33] Tong Jia, Pengfei Chen, Lin Yang, Ying Li, Fanjing Meng, Jingmin Xu, An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services, in: IEEE International Conference on Web Services, ICWS, IEEE, 2017, pp. 25–32.
[34] Pooja Aggarwal, Ajay Gupta, Prateeti Mohapatra, Seema Nagar, Atri Mandal, Qing Wang, Amit Paradkar, Localization of operational faults in cloud applications by mining causal dependencies in logs using golden signals, in: ICSOC, Springer International Publishing, 2021, pp. 137–149.
[35] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, Guangba Yu, SwissLog: Robust anomaly detection and localization for interleaved unstructured logs, IEEE Trans. Dependable Secur. Comput. 20 (4) (2023) 2762–2780.
[36] Alvaro Brandon, Marc Solé-Simó, Alberto Huélamo, David Solans, María Pérez, Victor Muntés-Mulero, Graph-based root cause analysis for service-oriented and microservice architectures, J. Syst. Softw. 159 (2019) 110432.
[37] JinJin Lin, Pengfei Chen, Zibin Zheng, Microscope: Pinpoint performance issues with causal graphs in micro-service environments, in: Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, Springer, 2018, pp. 3–20.
[38] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, Christina 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.
[39] Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, Dai Zhang, Zhenyu Zhu, Dan Pei, Robust failure diagnosis of microservice system through multimodal data, IEEE Trans. Serv. Comput. 16 (06) (2023) 3851–3864.
[40] Yingying Zhang, Zhengxiong Guan, Huajie Qian, Leili Xu, Hengbo Liu, Qingsong Wen, Liang Sun, Junwei Jiang, Lunting Fan, Min Ke, CloudRCA: A Root Cause Analysis Framework for Cloud Computing Platforms, CIKM '21, New York, NY, USA, 2021, pp. 4373–4382.
[41] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Michael R. Lyu, Eadro: An end-to-end troubleshooting framework for microservices on multi-source data, in: Proceedings of the 45th International Conference on Software Engineering, ICSE '23, IEEE Press, 2023, pp. 1750–1762.