Illuminating the Gray Zone: Non-intrusive Gray Failure Localization in Server Operating Systems

Shenglin Zhang
Nankai University & HL-IT
Tianjin, China

Yongxin Zhao
Nankai University
Tianjin, China

Xiao Xiong
Nankai University
Tianjin, China

Yongqian Sun*
Nankai University &
TKL-SEHCI
Tianjin, China

Xiaohui Nie
CNIC
Beijing, China

Jiacheng Zhang
Nankai University
Tianjin, China

Fenglai Wang
Huawei Technologies
Nanjing, China

Xian Zheng
Huawei Technologies
Nanjing, China

Yuzhi Zhang
Nankai University
Tianjin, China

Dan Pei†
Tsinghua University &
BNRist
Beijing, China

ABSTRACT

Timely localization of the root causes of gray failure is essential for maintaining the stability of the server OS. The previous intrusive gray failure localization methods usually require modifying the source code of applications, limiting their practical deployment. In this paper, we propose *GrayScope*, a method for non-intrusively localizing the root causes of gray failures based on the metric data in the server OS. Its core idea is to combine expert knowledge with causal learning techniques to capture more reliable inter-metric causal relationships. It then incorporates metric correlations and anomaly degrees, aiding in identifying potential root causes of gray failures. Additionally, it infers the gray failure propagation paths between metrics, providing interpretability and enhancing operators' efficiency in mitigating gray failures. We evaluate *GrayScope*'s performance based on 1241 injected gray failure cases and 135 ones from industrial experiments in Huawei. *GrayScope* achieves the *AC*@5 of 90% and interpretability accuracy of 81%, significantly outperforming popular root cause localization methods. Additionally, we have made the code publicly available to facilitate further research.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

Gray Failure, Server Operating System, Root Cause Localization, Causal Discovery

ACM Reference Format:
Shenglin Zhang, Yongxin Zhao, Xiao Xiong, Yongqian Sun, Xiaohui Nie, Jiacheng Zhang, Fenglai Wang, Xian Zheng, Yuzhi Zhang, and Dan Pei. 2024. Illuminating the Gray Zone: Non-intrusive Gray Failure Localization in Server Operating Systems. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3663529.3663834

1 INTRODUCTION

Servers support countless applications and services, serving as the core of large-scale data management and a key component in providing network services [18]. Server operating system (OS) acts as an intermediary between applications and the server hardware. They provide essential services such as resource allocation, scheduling, input/output processing, and security management, enabling applications to run efficiently and securely on hardware.

In server OS, the complex interactions between components can easily lead to performance or availability issues. A significant concern is gray failure, a critical state where the server OS is between healthy and unhealthy and can cause system instability or operational efficiency decrease [14, 15, 24]. Gray failures include severe performance degradation, random packet loss, flaky I/O, memory thrashing, capacity pressure, and non-fatal exceptions [15]. It has been shown that gray failures are the root cause of many catastrophic failures in the real world [24]. For instance, a gray failure causing high disk I/O wait times in the server OS can slow down database applications that rely heavily on disk operations. Gray failures occur frequently but are difficult to localize, making troubleshooting time-consuming and inaccurate.

Due to the lack of practical fine-grained diagnostic tools, it is labor-intensive to localize gray failures accurately, requiring trial-and-error troubleshooting by operators. We have investigated the server OS failure tickets at *Huawei Cloud* and found that the average

[*] Yongqian Sun is the corresponding author. Email: sunyongqian@nankai.edu.cn
[†] HL-IT, TKL-SEHCI, and BNRist are short for Haihe Laboratory of Information Technology Application Innovation, Tianjin Key Laboratory of Software Experience and Human Computer Interaction, and Beijing National Research Center for Information Science and Technology, respectively.

FSE Companion '24, July 15–19, 2024, Porto de Galinhas, Brazil
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0658-5/24/07
https://doi.org/10.1145/3663529.3663834

time from the occurrence of a gray failure to localizing its root cause is about eight hours, with some lasting several weeks. Such failures significantly impact the performance and availability of systems, yet research on root cause localization for gray failures is relatively scarce. Some intrusive methods rely on modifying the source code of applications (*e.g.*, Panomara [14] and OmegaGen [24]), limiting their practical deployment due to high modification costs and long localization cycles.

In server OS, metrics are quantifiable measures that provide insights into performance and operational status. These metrics are collected non-intrusively, ensuring the monitoring process does not interfere with the system's normal functioning or degrade its performance. Several metrics serve as measures of application performance, known as Key Performance Indicators (KPIs), *e.g.*, average response time. Anomalies on KPIs often signal potential gray failures [21] (hereinafter, we use "KPI" to denote the metrics of applications and apply "metric" to represent other types of metrics). Therefore, operators swiftly undertake mitigation measures to prevent further damage once a KPI exhibits anomalies. In recent years, a collection of metric-based root cause localization methods has been proposed for distributed systems, such as microservices, Web services [6, 20, 23, 27, 30, 32, 36, 37, 46]. These methods can be roughly divided into statistical, graph-based, and feature learning methods [38]. Statistical methods [23, 37], which use traditional statistical analysis methods for root cause localization, are easily affected by data noise. However, in the server OS, the varying workload and the complex interdependencies between components can generate significant noise in the metric data. This noise can obscure the statistical patterns these methods rely on, leading to inaccurate root cause localization. Feature learning methods [20, 46] that use machine learning techniques to learn feature models from metric data for root cause localization, often relying on many high-quality labeled cases. It presents a significant challenge in server OSes, where the specific manifestations of failures can vary widely across different configurations. As a result, neither statistical nor feature learning methods designed for distributed systems are appropriate in localizing the root cause of gray failures in server OS. Causality graph-based methods, which learn causal graphs to derive potential root cause lists, have demonstrated superior performance in real-world scenarios [27, 30, 32, 36]. They are promising for non-intrusive metric-based gray failure localization in server OS.

Based on the observations above, our objective is to design a causality graph-based method to localize the root causes of gray failures in server OS non-intrusively, accurately, and timely. However, it faces the following challenges:

(1) Complex causal relationships between metrics. Usually, there are hundreds of metrics in a server OS. The metric data changes dynamically in the server OS over time, and the relationships between these metrics also evolve dynamically. Previous works apply only causal learning techniques, such as the PC algorithm [39] or the Granger causality test [10], to construct the causal graph, making the causal graph too large and complex for root cause inference.

(2) Underutilization of the correlations. When a gray failure occurs, usually, some metrics become anomalous. The anomaly degree of metrics helps prioritize exploration in the root cause inference, focusing on metrics potentially involved in the gray failure. Previous works usually utilize root cause inference techniques like random walk [34] or Page Rank [1] by considering only the anomaly degree of metrics while ignoring the correlations between each metric and the gray failure. However, the correlation between metrics and the gray failure can guide the root cause inference method to localize the metrics causing the gray failure. Ignoring this correlation can degrade the performance of root cause inference.

(3) Interpretability. Gray failures in system components can spread with data flow or shared resources, potentially expanding the impact on the entire system. A lack of information about the propagation paths of gray failures can affect the efficiency of operators in mitigating failures.

In this paper, we propose a novel method, *GrayScope*, tailored for gray failures in server OS and can non-intrusively, timely, and accurately localize the root cause of gray failures. The main contributions of this paper are as follows:

(1) To the best of our knowledge, *GrayScope* is the first work targeting root cause localization of gray failures for server OS in a non-intrusive manner. It significantly improves operators' efficiency in diagnosing and mitigating failures.

(2) To learn more reliable causal relationships between metrics, *GrayScope* integrates expert knowledge with causal learning techniques to learn the metric causality graph, addressing the first challenge. Moreover, it combines partial correlation with anomaly degree to localize root causes, addressing the second challenge. In addition, we present a simple yet effective failure propagation path inference technique to infer the gray failure propagation paths between metrics, addressing the third challenge.

(3) To evaluate the performance of *GrayScope*, we collect the data of 1241 gray failures from 16 different server OSes in *Huawei*. Our results show that *GrayScope* achieves the top 5 accuracy ($AC@5$) of 90% and the average top 5 accuracy ($Avg@5$) of 82%, significantly outperforming advanced failure localization methods designed for distributed systems. To ensure better reproducibility, we have made our code publicly available [11]. We also validate *GrayScope*'s performance using gray failure cases collected from the industrial environment.

2 BACKGROUND

2.1 Preliminaries

2.1.1 Gray Failure. When at least one component perceives the server OS as unhealthy while observers observe the server OS as healthy, the server OS is experiencing a gray failure [14, 15, 24]. Gray failures include performance degradation, capacity pressure, flaky I/O, memory thrashing, random packet loss, and non-fatal exceptions. They lead to a critical state where the server OS is between healthy and unhealthy states. The server OS can still deliver some normal functions but operates in a mode of decreased performance or degraded functionality.

2.1.2 KPI and Metric. The assessment of application performance and server OS state frequently relies on two principal categories of indicators: Key Performance Indicator (KPI) and metric [30]. KPIs (*e.g.*, average response time, error rate, and page view count) are indicators used to assess and monitor the performance of applications. They provide valuable information about the operational state of applications, user experience, and system efficiency. Metrics

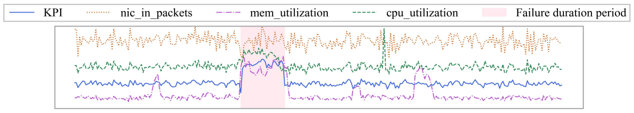


Figure 1: The KPI and some metrics in a server OS during a gray failure caused by CPU exhaustion.

(*e.g.*, CPU utilization, memory utilization, and network throughput), on the other hand, indicate the status of the server OS's foundational components. An anomalous metric, suggesting an anomaly in a server OS, can potentially cause a KPI anomaly. However, not all metric anomalies cause KPI to be anomalous, as minor or transient fluctuations in server OS component performance do not necessarily translate to a degradation in application performance.

In server OS, gray failures are often accompanied by anomalies in KPIs [15]. To quickly mitigate these gray failures, it is essential to localize the root cause of the gray failure upon detecting anomalies in KPIs. KPI is variant in different applications [4]. In this work, the application scenarios we study mainly include GaussDB [16] (an enterprise-grade distributed relational database from *Huawei*), Redis [5], Kafka [47], and Tomcat [29]. With the domain knowledge of operators, we choose the database throughput as the KPI for GaussDB, the database request response time for Redis, the message production rate per second for Kafka, and the Servlet request processing time for Tomcat.

2.1.3 Problem Statement. Our objective is to design an accurate and efficient method for identifying root cause metrics to assist operators in quickly repairing gray failures in server OS. Specifically, when a gray failure occurs in a server OS, and an application's KPI is detected as anomalous, we collect all relevant metrics from that server OS. We aim to provide operators with a ranked list of root cause metrics and the possible gray failure propagation paths for each potential root cause. Given that an anomaly of a component can propagate through the server OS [9], and multiple metrics may exhibit anomalies during a gray failure [51], localizing the root causes (recognizing a small set of root cause metrics) is crucial for mitigating gray failures as operators can take targeted mitigation measures to prevent recurrence, optimize system performance. For instance, consider a scenario involving CPU exhaustion gray failure, where the identified root cause metric is CPU utilization. This metric indicates the CPU is overused, leading to system slowdowns or performance degradation. This information allows operators to investigate processes or applications that consume excessive CPU resources. Note that gray failure detection is beyond the scope of this paper.

2.2 Motivation

We obtain a collection of gray failure cases from *Huawei*'s server OSes. A common symptom observed across all cases is the degradation or interruption of application performance. Once a KPI is anomalous, operators will act swiftly to mitigate the gray failure to prevent further damage. Analyzing these cases provided insights for designing a gray failure localization framework. Due to confidentiality reasons, we have to hide the details of these cases.

Gray failures are common and can propagate in server OS. Gray failures typically exhibit a typical evolution pattern along the temporal dimension: Initially, the system may experience minor failures (latent failures) that are often suppressed. Gradually, the system transits to partial failure, where some but not all functionalities are compromised (gray failures). Ultimately, the continued spread of partial failures may lead to a system crash (complete failures) [15]. Gray failures are the root of many catastrophic failures [24]. They can cause severe damage, including inconsistencies, "zombie" behavior, and data loss, which operators cannot ignore.

Given that modern server OS consists of many highly interactive components across layers, when one component becomes unhealthy, it will likely impact the performance of other components (possibly all), affecting the system's regular operation. Consequently, timely and accurate localization and mitigation of gray failures in server OS are crucial for ensuring their high availability. However, current research on the root cause localization of gray failures in server OS is scarce. The root cause localization methods designed for failures in distributed systems might be ineffective for server OS gray failures due to limitations of the methods (such as the inability to learn dependencies accurately, the need for manual labels, etc.). Therefore, we propose *GrayScope* for localizing the root causes of gray failures in server OS.

Expert knowledge is essential for accurate causality learning. Identifying causal relationships is a critical task that involves the intersection of causality and machine learning [44]. Previous research has successfully explored causal relationships within time series data through diverse methodologies [27, 30, 32, 36]. Nevertheless, these studies frequently overlook the valuable expert knowledge on causality, deeply rooted in the extensive experience of operators who have managed numerous system failures. It leads to uncertain or inaccurate causality, which degrades the performance of the gray failure localization process. Incorporating engineers' rich diagnostic experience can reduce this uncertainty and enhance causality accuracy [2, 12, 44].

Our examination of eight gray failure cases in server OS collected from *Huawei* determined that the fusion of expert knowledge is crucial in identifying the root causes of gray failures. Specifically, we use three popular methods, including Granger causality tests [10], PC algorithm [39], and PCTS algorithm [30], to construct metric causality graphs. By incorporating expert knowledge, we corrected erroneous edges. Subsequently, we employ the Root Cause Inference module (see Section 3.4) of *GrayScope* to infer the root causes of gray failures. As listed in Table 1, the results underscore the critical importance of expert knowledge in accurately localizing gray failure root causes. When constructing causality graphs using various methods, fusing expert knowledge corrects some erroneously learned causal relationships. Moreover, after integrating expert knowledge, there is a noticeable improvement in the accuracy of root cause localization.

Root cause metric exhibits anomaly during a gray failure and correlates with the KPI. Since applications often rely on various underlying components, metrics reflecting the health or performance of these components directly relate to the application's overall performance. For example, high CPU utilization can lead to extended response times for processing requests. Similarly, excessive disk utilization or memory leaks can affect application performance. As shown in Fig. 1, the KPI exhibited anomaly and represented a gray failure in server OS. The root cause of this

Table 1: The number of edges in the causality graph constructed by different methods and the results of gray failure localization (✓for accurate localization and × for inaccurate localization).

Method	Disk Failure_1	Disk Failure_2	Delay Failure_1	Delay Failure_2	Packet Loss Failure_1	Packet Loss Failure_2	CPU Failure_1	CPU Failure_2
Granger causality tests [10] w knowledge	76 (✓)	92 (✓)	88 (✓)	81 (✓)	42 (✓)	142 (✓)	63 (✓)	54 (✓)
Granger causality tests [10] w/o knowledge	297 (×)	345 (×)	152 (✓)	153 (✓)	155 (×)	395 (×)	210 (✓)	217 (×)
PC algorithm [39] w knowledge	12 (×)	42 (✓)	7 (×)	6 (×)	16 (✓)	15 (×)	31 (✓)	3 (×)
PC algorithm [39] w/o knowledge	59 (×)	95 (×)	40 (×)	43 (×)	54 (✓)	64 (×)	60 (×)	53 (×)
PCTS algorithm [30] w knowledge	32 (✓)	47 (✓)	52 (✓)	50 (×)	48 (✓)	45 (✓)	64 (✓)	43 (×)
PCTS algorithm [30] w/o knowledge	40 (✓)	51 (×)	69 (✓)	63 (×)	73 (✓)	48 (✓)	64 (✓)	89 (×)

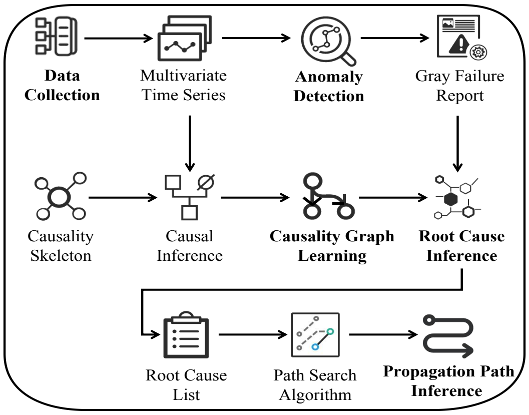


Figure 2: The framework of *GrayScope*.

gray failure was high CPU utilization, which led to extended response times for processing. During the gray failure, the metrics "mem_utilization" and "cpu_utilization" also showed anomalies, while the metric "nic_in_packets" appeared normal. Although the anomaly degree of "mem_utilization" is higher, "cpu_utilization" correlates more with the KPI when the gray failure occurred. As a result, the correlation between metrics and the KPI, as well as the anomaly degree of metrics, are crucial for accurately identifying the root cause of the gray failure.

3 APPROACH

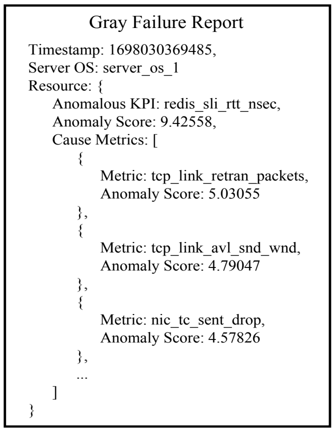
3.1 Overview

As shown in Fig. 2, *GrayScope* consists of four key modules:
(1) **Data Collection and Anomaly Detection (§3.2).** *GrayScope* uses metric collection tools to collect runtime monitoring metrics from multiple data sources at fixed intervals in server OS. It then triggers root cause localization when an anomaly in KPI is detected.
(2) **Causality Graph Learning (§3.3).** In root cause localization, *GrayScope* first constructs a metric causality structure graph by plugging relevant metrics in a skeleton graph based on expert knowledge. It then analyzes causal relationships between metrics using an observation window for causality testing. By integrating the metric causality structure graph with the causal relationships between metrics, a metric causality graph is derived, representing how various metrics affect KPI and their mutual interactions. Therefore, *GrayScope* focuses on relevant metrics rather than all available data, reducing the chance of spurious correlations. The derived metric causality graph captures the evolving causal structure, considering direct and indirect relationships between metrics and the target KPI, overcoming the challenge introduced by complex causal relationships between metrics (the first challenge).
(3) **Root Cause Inference (§3.4).** Inspired by the random walk algorithm, *GrayScope* considers the weighted combination of the correlation between metrics and KPI, as well as the anomaly degree of metrics themselves, as the transition probabilities for the walk. Starting from the anomalous KPI, *GrayScope* randomly traverses along the metric causality graph to generate a potential root cause ranking list. Integrating the correlation provides a dynamic assessment of potential root causes, mitigating the underutilization of the correlations (the second challenge).
(4) **Propagation Path Inference (§3.5).** Finally, *GrayScope* combines the metric causality graph with the potential root cause ranking list to infer possible propagation paths of the root cause within the server OS. This provides an interpretable explanation of how the gray failure spreads through the server OS, aiding operators in implementing targeted mitigation strategies, thus addressing the challenge of interpretability (the third challenge).

3.2 Data Collection and Anomaly Detection

By real-time monitoring of server OS metrics, operators can promptly identify and resolve performance issues, ensuring that server OS meets user requirements. The Data Collection module gathers multiple runtime information from the server OS across various data sources, including system calls, applications, and process communications. Gala-gopher [8] is an eBPF-based low-overhead probe framework for monitoring and collecting data on server OS network, memory, disk I/O, and scheduling states. It allows for configuring existing collection probes based on business needs. We deploy gala-gopher on each server OS to collect monitoring metrics, setting the data collection interval at five seconds. Prometheus [35] is an open-source service monitoring system and time-series database, providing a generic data model and fast data collection, storage, and query interfaces. We employ Prometheus to collect metric data from each server OS at given intervals.

As described in Section §2.1.3, gray failures, when they occur, can lead to a degradation in application performance, with anomalies in KPI. Therefore, before localizing the root cause of gray failures, we first require an anomaly detection algorithm to identify anomalies in KPI and report the gray failure occurring in the system. Furthermore, according to Pearl's concept of cause-effect [33], if there is a causal relationship between two variables, a change in one variable will lead to a change in the other. Usually, the root cause metrics will also exhibit anomalies during the gray failure, as anomalous metrics could be the potential root causes of an abnormal KPI [30].



```
Gray Failure Report
Timestamp: 1698030369485,
Server OS: server_os_1
Resource: {
    Anomalous KPI: redis_sli_rtt_nsec,
    Anomaly Score: 9.42558,
    Cause Metrics: [
        {
            Metric: tcp_link_retran_packets,
            Anomaly Score: 5.03055
        },
        {
            Metric: tcp_link_avl_snd_wnd,
            Anomaly Score: 4.79047
        },
        {
            Metric: nic_tc_sent_drop,
            Anomaly Score: 4.57826
        },
        ...
    ]
}
```

Figure 3: An example of gray failure report.

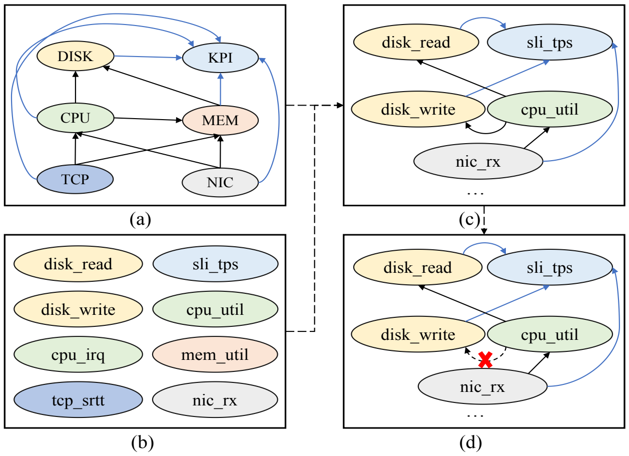


Figure 4: Construction of the causality graph in *GrayScope*. (a): meta-metric causality skeleton graph; (b): monitoring metrics plugging-in; (c): metric causality structure Graph; (d): metric causality graph.

We use gala-anteater [7] to detect anomalies in metrics and KPIs, as it integrates various time series anomaly detection algorithms, enabling real-time anomaly detection for different scenarios and applications. Additionally, it provides anomaly scores for each metric, which provides crucial input for the downstream Root Cause Inference module. Specifically, it collects data in real-time from Prometheus and outputs the moment when the anomaly has occurred, the anomalous KPI, and the anomaly scores for all metrics. Finally, it generates a gray failure report, as shown in Fig. 3.

3.3 Causality Graph Learning

Previous research [6, 27, 30, 32, 36] has shown that learning effective causality graphs is crucial for failure root cause localization. Numerous studies have utilized the PC algorithm [39] to learn causality graphs between metrics. However, the PC algorithm can only learn instantaneous causal relationships and fails to report the extensive continuous causal relationships between time series [30]. Other works [31, 32, 40, 45] have employed Granger causality tests [10], a method of time series analysis used to test for causality between two time series, to learn causality graphs between metrics. Its basic idea is that if the past values of time series X can better predict the current value of another time series Y, and Y’s past values do not provide a better prediction for X’s current value, then we can say X Granger-causes Y. However, in our scenario, it results in numerous false causal relationships, leading to low accuracy in root cause localization. Therefore, we propose a causality graph learning model that combines expert knowledge with Granger causality tests.

Based on the data sources of the collected metrics, we categorize the monitoring metrics into six dimensions: performance-related metrics, CPU-related metrics, memory-related metrics, network-related metrics, disk-related metrics, and TCP-related metrics, which we refer to as meta metrics. The causal relationships between these six categories of meta-metrics serve as a benchmark for causality testing between metrics. As shown in Fig. 4(a), leveraging expert knowledge of gray failure patterns, we construct a causality skeleton graph of meta metrics for server OS gray failures. It is worth noting that expert knowledge is considered a one-shot cost. *GrayScope* enables users to categorize data gathered based on specific application scenarios and draw upon historical instances of failures to establish the causality skeleton graph. This method not only diminishes the continual reliance on expert intervention but also augments the adaptability and scalability of *GrayScope* in response to evolving scenarios.

We assume metric X is not a Granger cause of Y. We establish a multivariate regression model with an appropriate *maxlag* and perform an F-statistical test to compare the fit of models with and without X’s lagged terms. We determine whether to reject the null hypothesis based on the F-statistic and significance level $p\_threshold$. If the null hypothesis is rejected, it indicates that variable X Granger causes variable Y.

As shown in Fig. 4(a), in the expert-knowledge-based causality skeleton graph constructed, we insert the top m related metrics with the highest anomaly scores for each category of meta-metrics into the respective types as shown in Fig. 4(b), resulting in the metric causality structure graph, as shown in Fig. 4(c). Specifically, in the causality skeleton graph, the network meta-metrics point towards the CPU meta-metrics, suggesting that, based on expert knowledge, network-related metrics might be the root cause of CPU-related metrics. After plugging each meta-metric, their corresponding related metrics maintain full connectivity, meaning all network-related metrics point towards every CPU-related metric. However, causal relationships between two time series do not always exist [32]. Therefore, we utilize the w-minute observation window data before the gray failure report time for testing. We perform the Granger causality test for all related metrics using time series data from the most recent w-minute window obtained from gala-gopher. If the test confirms a causal relationship, we retain the current edge; otherwise, we remove it. Finally, we preserve the subgraph containing the anomalous KPI, resulting in the learned metric causality graph, as shown in Fig. 4(d).

3.4 Root Cause Inference

Random walk algorithms are based on randomness and are commonly used to simulate random graph traversal. They inspire our proposed root cause inference algorithm. Identifying root causes should prioritize metrics highly correlated with KPI [27]. Additionally, root cause metrics usually exhibit anomalies during a gray failure [30]. Therefore, the random walk should consider the correlation between each metric and the anomalous KPI and each metric's anomaly degree.

To infer the root cause, we apply a random walk on the obtained causality graph. Specifically, we assume a transition probability matrix H, where each element in H represents the transition probability between any two metrics. Suppose there are metrics v_i and v_j in the causality graph, with an edge from v_j to v_i, the transition probability between v_j and v_i is calculated as follows:

First, compute the partial correlation coefficient $correlation(v_j)$ between v_j and the KPI. Then, calculate the relative anomaly degree of v_j using the anomaly score:

$$anomaly\_degree(v_j) = \frac{anomaly\_score(v_j)}{anomaly\_score(v_i) + anomaly\_score(v_j)} \tag{1}$$

The transition probability matrix H between metrics is obtained by weighting the partial correlation coefficients and relative anomaly degrees. Given the causality graph and the transition probability matrix H, the visitor starts from v_{KPI}, calculating the probabilities for forward, backward, and self-transitions, and randomly walks along the graph. We calculate the matrix H' in random walk as follows:

(1) Forward step (walk from result metric to cause metric):

$$H'_{i,j} = \lambda \cdot correlation(v_j) + (1 - \lambda) \cdot anomaly\_degree(v_j) \tag{2}$$

where $\lambda \in [0, 1]$ is the weight that controls the contribution of partial correlation coefficient with the anomalous KPI and the anomaly degree of the metric.

(2) Backward step (walk from cause metric to result metric):

$$H'_{j,i} = \rho \cdot (\lambda \cdot correlation(v_i) + (1 - \lambda) \cdot anomaly\_degree(v_i)) \tag{3}$$

where ρ is a parameter controlling the impact of the backward step and $\rho \in [0, 1]$.

(3) Self step (stay in the present metric):

$$H'_{j,j} = \max[0, H'_{j,j} - H'^{max}_{j,k}] \tag{4}$$

$$H'^{max}_{j,k} = \max H'_{j,k} \tag{5}$$

where v_k is a neighboring metric of v_j. If the algorithm reaches a metric and the probability of transitioning to a neighboring metric is low, then this metric is likely the root cause, and we consider that the walker should stay at this metric.

Finally, we get H by normalizing every row of H'. The next metric in the sequence of adjacent metrics is randomly selected for visiting. The probability of visiting each metric is proportional to its anomaly degree and correlation with the anomalous KPI. We record the times each metric is visited and arrange them in descending order as the root cause localization result.

3.5 Propagation Path Inference

Localizing gray failures is crucial for rapidly taking necessary measures to mitigate them. Studying the propagation paths of gray failures can assist operators in identifying critical metrics along the gray failure propagation path, improve their confidence about the gray failure localization result, shorten the mitigation time of the gray failure, and enhance system availability.

Our goal is to deduce the gray failure propagation path from v_{root} to v_{KPI}, based on the current anomalous KPI v_{KPI} obtained from the Anomaly Detection module and the potential root cause metric v_{root} from the Root Cause Inference module. To achieve this, we designed a method that combines the metric causality graph with each metric's anomaly score, the anomalous KPI v_{KPI}, and the root cause metric v_{root} to infer potential propagation paths, as shown in Algorithm 1. We aim to find the shortest path with the metrics' highest cumulative anomaly score as the most likely gray failure propagation path [36].

Algorithm 1: *GrayScope* Gray Failure Propagation Path Inference

```
Input: anomalous KPI: $v_{KPI}$; root cause metric: $v_{root}$; metrics'
       anomaly score: $AS$; metric causality graph: $G$
Output: gray failure propagation $path$ from $v\_root$ to $v\_KPI$
$G'$ ← construct the undirected graph based on $G$
$V$ ← vertices set based on $G'$
$E$ ← edge set based on $G'$
$w_{i,j}$ ← 0 // weight of edge $e_{i,j} \in E$
foreach edge $e_{i,j} \in E$ do
    $w_{i,j} = \frac{2}{AS[i]+AS[j]}$
end
foreach vertex $v \in V$ do
    $dist[v] = \infty$ // Initial distance
    $prev[v] = UNDEFINED$ // Previous vertex
end
$dist[v_{root}] = 0$ // Distance from source to source
$Q = \emptyset$ // Priority queue to hold vertices
foreach vertex $v \in V$ do
    // Add each vertex to the priority queue
    $Q$.ADD-WITH-PRIORITY($v$, $dist[v]$)
end
while $Q \neq \emptyset$ do
    // Extract vertex with min distance
    $u = Q$.EXTRACT-MIN()
    if $u == v_{KPI}$ then
        // Break if destination vertex is reached
        break
    end
    foreach neighbor $v$ of $u$ do
        $alt = dist[u] + w_{u,v}$
        if $alt < dist[v]$ then
            $dist[v] = alt$
            $prev[v] = u$
            // Update priority in the queue
            $Q$.DECREASE-PRIORITY($v$, $alt$)
        end
    end
end
return path from $v_{root}$ to $v_{KPI}$ using $prev[\,]$
```

4 EVALUATION

We aim to answer the following research questions (RQs):

Table 2: Dataset information

Dataset	#CPU Exhaustion	#Disk IO High Load	#Network Latency	#Network Packet Loss
GaussDB	0	78	62	83
Redis	0	196	46	32
Kafka	20	0	94	187
Tomcat	192	0	134	117

RQ1: How effective is *GrayScope* in root cause localization?
RQ2: Does each component of *GrayScope* contribute significantly to *GrayScope*'s performance?
RQ3: How accurate is *GrayScope* in inferring propagation paths?
RQ4: What is the impact of different hyperparameters?

4.1 Experimental Setup

Dataset. To comprehensively evaluate the performance of *GrayScope*, we establish a cluster environment in *Huawei*, comprising five physical host machines and 11 virtual machines. EulerOS, a Linux distribution developed by *Huawei* based on Red Hat Enterprise Linux to provide OS for server and cloud environment [52], is installed on each of these 16 machines, and extensive experiments are conducted on them. Four popular applications are deployed across these server OSes to ensure comprehensive experimentation. We use the chaos engineering experimental tool Chaosblade [3] for gray failure simulation to simulate network latency, packet loss, disk IO high load, and CPU exhaustion. In our experiments, each gray failure simulation lasts for two minutes, with an interval of approximately 20 minutes between injection operations to minimize the interaction effects between different gray failures. We inject 1241 gray failures, including 212 gray failures caused by CPU exhaustion, 274 caused by disk IO high load, 336 caused by network latency, and 419 caused by network packet loss. Table 2 lists the detailed information about the dataset. Various types of gray failures may introduce performance degradation for different applications; hence, different types of gray failures are injected into the server OS hosting different applications. Specifically, CPU exhaustion does not affect the performance of GaussDB or Redis applications. Therefore, the gray failures caused by CPU exhaustion are not injected in these two applications. Similarly, the gray failures caused by disk IO high load do not impact the performance of Kafka or Tomcat applications, so these are not injected in these two application scenarios.

Implementation. *GrayScope* is implemented using Python 3.8. We have made the source code publically available [11]. All of the experiments are conducted on each server OS. As for the hyperparameters, in the Granger causality test, the significance level threshold *p_threshold* and the maximum lag order *maxlag* are set to 0.05 and 2, respectively. The parameter ρ, controlling the influence of the backward steps in the random walk, is set to 0.2. The parameter *corr_prop*, which controls the weights of the partial correlation coefficient and the anomaly degree of metrics, is set to 0.2. We will discuss these parameters in Section 4.5.

Baselines. We select the following approaches as baselines because these approaches use different types of techniques to localize the root causes of gray failures at the metric level, which aligns with the goal of *GrayScope*: (1) CauseInfer [4] constructs causality graphs based on the PC algorithm, uses a DFS strategy to traverse the causality graph, and infers potential root cause metrics based on anomaly scores. We set the sliding window length $w = 36$. (2) MicroCause [30] applies a path condition time series (PCTS) algorithm to learn the dependency graph of metrics and employs a temporal cause-oriented random walk (TCORW) algorithm to rank root cause metrics. We set $\rho = 0.5$ and $\lambda = 0.1$, where ρ is a parameter controlling the impact of the backward step and λ controls the contribution of metric's causal relationship with the anomalous KPI and the anomaly degree of the metric. (3) TS-InvarNet [13] constructs an invariant network by modeling each pair of time series. The parameter lag order *Lags* is set to 3. (4) CIRCA [19] constructs the skeleton based on system architecture and uses regression-based hypothesis testing (RHT) and descendant adjustment methods to infer failure root cause metrics in the graph. Its training and testing window lengths are set to 110 and 10, respectively.

Evaluation metrics. To evaluate *GrayScope* and baseline methods, we use *AC@k* and *Avg@k* to assess their outcomes. These metrics are the most commonly used in the literature [19, 30, 50]. *AC@k* represents the probability that the top k results given by each method include the real root causes of all given gray failure cases. A higher *AC@k* value, especially for smaller k, indicates greater method accuracy in identifying root causes. Since the search space is smaller, the methods with higher *AC@k* can significantly improve the efficiency of operators in troubleshooting gray failures. Given a set of gray failure cases A, *AC@k* is calculated as follows:

$$AC@k = \frac{1}{|A|} \sum_{a \in A} \frac{\sum_{i \le k} R_a\left[i\right] \in V_a}{min(k, |V_a|)} \tag{6}$$

where $R_a\ [i]$ is the ranking result of all metrics for a gray failure case a, and V_a is the actual root cause set for the gray failure case a. *Avg@k* evaluates the overall performance of each method by calculating the average *AC@k*. *Avg@k* is calculated as follows:

$$Avg@k = \frac{1}{k} \sum_{1 \le j \le k} AC@j \tag{7}$$

4.2 Overall Performance (RQ1)

Table 3 lists the performance of different methods across four application scenarios. The complexity is analyzed by comparing the time required to localize each gray failure case. It takes 8.74 s for *GrayScope*, 307.72 s for CauseInfer [4], 19.71 s for MicroCause [30], 9.96 s for TS-InvarNet [13], and 2.40 s for CIRCA [19]. *GrayScope* achieves a satisfactory gray failure localization efficiency. *GrayScope* demonstrates superior performance compared to all baseline methods, achieving an impressive *AC*@5 of 0.90. Specifically, *GrayScope*'s accuracy at *AC*@5 is 10% higher than that of TS-InvarNet [13], the best-performing baseline method. Regarding overall performance, *GrayScope* outperforms others, with its *Avg*@5 reaching 0.82, a relative improvement of 18% compared with the second place. Integrating expert knowledge for capturing more accurate causal relationships and combining partial correlation with anomaly degree for root cause inference empowers *GrayScope*'s success.

Except for CIRCA [19], all other baseline methods ignore the significance of expert experience, which is essential for accurately identifying the root causes of gray failures in server OS. However,

Table 3: Effectiveness of root cause localization

Method	All			GaussDB			Redis			Kafka			Tomcat		
	AC@3	*AC@5*	*Avg@5*	*AC@3*	*AC@5*	*Avg@5*	*AC@3*	*AC@5*	*Avg@5*	*AC@3*	*AC@5*	*Avg@5*	*AC@3*	*AC@5*	*Avg@5*
GrayScope	**0.86**	**0.90**	**0.82**	**0.96**	**0.97**	**0.95**	**0.97**	**0.97**	**0.91**	**0.81**	**0.85**	**0.80**	**0.77**	**0.86**	**0.70**
CauseInfer [4]	0.23	0.25	0.21	0.39	0.41	0.37	0.42	0.49	0.40	0.14	0.15	0.12	0.09	0.10	0.08
MicroCause [30]	0.68	0.75	0.64	0.69	0.73	0.67	0.75	0.84	0.69	0.57	0.63	0.55	0.71	0.79	0.65
TS-InvarNet [13]	0.68	0.80	0.63	0.87	0.93	0.81	0.86	0.93	0.81	0.49	0.66	0.46	0.60	0.74	0.55
CIRCA [19]	0.51	0.64	0.50	0.74	0.83	0.73	0.92	0.95	0.88	0.39	0.57	0.38	0.21	0.39	0.22
C1	0.71	0.82	0.66	0.89	0.93	0.83	0.85	0.92	0.79	0.55	0.73	0.50	0.65	0.76	0.59
C2	0.27	0.34	0.26	0.47	0.54	0.45	0.42	0.47	0.40	0.18	0.24	0.17	0.15	0.21	0.15
C3	0.58	0.74	0.59	0.64	0.68	0.64	0.71	0.77	0.68	0.46	0.73	0.53	0.55	0.76	0.55
C4	0.73	0.80	0.69	0.70	0.75	0.69	0.84	0.88	0.78	0.68	0.73	0.67	0.71	0.81	0.66
C5	0.54	0.73	0.52	0.74	0.96	0.70	0.88	0.95	0.82	0.31	0.51	0.31	0.38	0.64	0.38

CIRCA [19] localizes root causes based on metric data distribution, which is easily impacted by the data noise in server OS. The unsatisfactory performance of CauseInfer [4] is attributed to its use of the PC algorithm, which fails to learn continuous causal relationships in time series, and its reliance on DFS for root cause inference based solely on whether metrics are anomalous, leading to inaccuracy. MicroCause's [30] reduced accuracy is due to learning some incorrect metric causality relationships. TS-InvarNet [13] localizes root causes based on the out-degree size of nodes in the causality graph, neglecting the failure propagation process across the entire causal network. Its inference results are limited as they consider only local features surrounding the nodes.

4.3 Contribution of Key Components (RQ2)

To demonstrate the effectiveness of different critical components in *GrayScope* (causality graph construction; root cause inference), we reconfigured *GrayScope* by removing or substituting its components with standard and state-of-the-art techniques, creating five variants, C1-C5. (1) **C1** removes the skeleton graph from *GrayScope* and only uses causal inference between metrics (Granger causality test) to build the causality graph. (2) **C2** uses the PC algorithm instead of the Granger causality test and combines it with a skeleton graph based on expert knowledge for causality graph construction. (3) **C3** relies solely on partial correlation coefficients for root cause inference. (4) **C4** bases root cause inference only on the anomaly degrees of metrics. (5) **C5** uses DFS instead of the random walk for root cause localization.

Table 3 lists *GrayScope*'s superior performance across various application scenarios compared to all the variants mentioned above, demonstrating each component's significance. When the skeleton graph is removed **(C1)**, both AC@5 and Avg@5 degrade. The results indicate that the skeleton graph can capture causal relationships between different metrics, thereby accurately learning inter-metric causal relationships by combining causal learning techniques. AC@5 and Avg@5 decrease when the PC algorithm replaces the Granger causality test **(C2)**, suggesting that the Granger causality test is more effective in learning the continuous causal relationships in time series than the PC algorithm. The performance becomes worse when relying solely on partial correlation coefficients or solely on the anomaly degrees of metrics **(C3 & C4)**. The reason is that both partial correlation coefficients and anomaly degrees of metrics can provide effective assistance for root cause inference, and both should be considered simultaneously during the inference process. When DFS replaces the random walk **(C5)**, the performance degrades as judging whether a metric is the root cause solely based on whether the metric is anomalous is not enough.

4.4 Interpretability (RQ3)

To the best of our knowledge, no standard in the academic or industrial community can definitively determine the accuracy of a given failure propagation path. Accordingly, we conducted a random sampling of 200 cases wherein *GrayScope* accurately identified the root cause at the top 1 position. Following this, we invited the expertise of two seasoned operators, each possessing a minimum of five years of operational experience, to assess the potential gray failure propagation paths delineated by *GrayScope* in these instances. Their task was to ascertain the correctness or incorrectness of these paths. When discrepancies arose between the assessments provided by the two operators, a third experienced operator was engaged to adjudicate, thereby ensuring the fidelity and impartiality of the evaluation.

The results indicated that out of the sampled 200 cases, *GrayScope* correctly identified the gray failure propagation path in 163 cases. We reasonably infer that *GrayScope* can accurately provide the correct gray failure propagation path with an accuracy rate of 81.5%, which assists operators in rapidly mitigating the gray failures.

4.5 Hyperparameters Sensitivity (RQ4)

We will discuss the impact of four hyperparameters of *GrayScope*. Figure 5 shows the variation of AC@5 with different hyperparameter settings.

(1) The significance level threshold. 0.05 is an ordinary significance level, and we aim to ensure that our conclusions are statistically significant. Hence, $p\_threshold$ is usually set to less than 0.05. When $p\_threshold$ changes from 0.01 to 0.05, the performance of *GrayScope* remains relatively stable. Setting the $p\_threshold$ to the expected value of 0.05 is more reasonable for determining the existence of causality, resulting in better model performance.

(2) Maximum lag order. This parameter represents the maximum lag order in the Granger causality test. *GrayScope* performs optimally when $maxlag$ is set to 2. When $maxlag$ varies from 1 to 5, the *AC*@5 of *GrayScope* remains almost unchanged. The experiment shows that *GrayScope* is insensitive to changes in $maxlag$.

(3) Parameter controlling the impact of a backward step in the random walk. Here, we change ρ from 0 to 1, and the results show that when ρ is greater than 0, the *AC*@5 of *GrayScope* tends to stabilize. *GrayScope* performs optimally when ρ is set to 0.2.

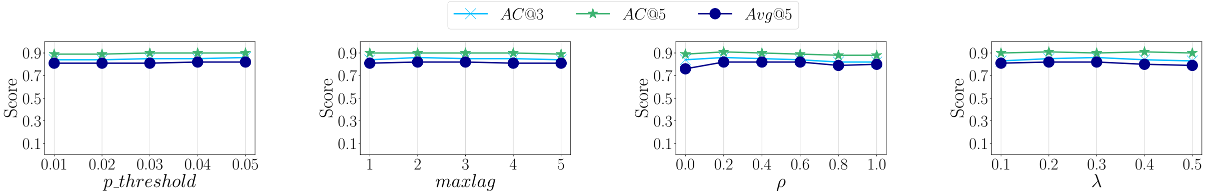


Figure 5: Parameters sensitivity on *GrayScope*.

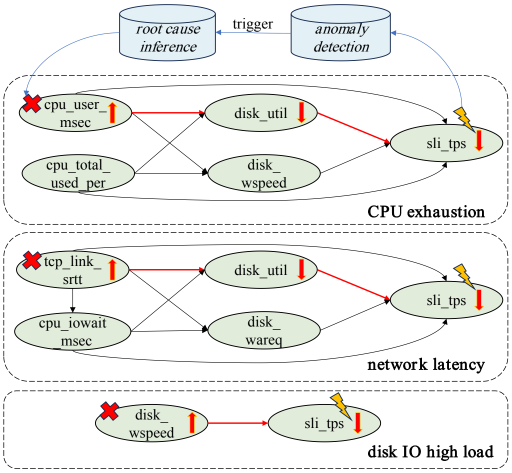


Figure 6: Gray failure cases: the red arrows represent the gray failure propagation path.

(4) Weight of partial correlation coefficient and anomaly degree. As can also be seen in the experiments in Section §4.3, the anomaly degree has a more significant impact on the accuracy of the results than the partial correlation coefficient. Therefore, we set λ to be less than or equal to 0.5. When λ changes from 0.1 to 0.5, the *AC*@5 of *GrayScope* remains almost unchanged. Overall, *GrayScope* works best when λ is set to 0.2.

4.6 Threats to Validity

A primary threat to the internal validity of our study lies in the implementation process of *GrayScope*. Our implementation is based on mature frameworks to mitigate this potential threat and has undergone rigorous checks and testing.

Regarding the external validity of our research, a potential threat lies in our study subjects. All our studies use data collected from four application scenarios in *Huawei*'s server OS. However, we believe that our approach possesses sufficient generality. Server OS business scenarios in other companies or domains may have different characteristics, such as metric fluctuations and anomaly propagation. Our study's accuracy and efficiency might not directly apply to other application scenarios. In the future, *GrayScope* will include further evaluations to make it applicable to a broader range of application scenarios.

5 DISCUSSION

5.1 Case Study

GrayScope has been deployed in *Huawei Cloud* for four months to help operators timely and accurately localize gray failures for server OSes. *GrayScope* is deployed on each server OS and is triggered when a KPI becomes anomalous. We further evaluate the performance of *GrayScope* based on a dataset collected from the industrial environment of *Huawei Cloud*, denoted as $\boldsymbol{C}$. There are 135 server OS gray failure cases in $\boldsymbol{C}$. Among 48 cases caused by network latency, *GrayScope*'s AC@3 reached 0.83; in 50 cases caused by disk IO high load, the AC@3 achieved 0.98; and among 37 cases caused by high memory utilization, the AC@3 attained 0.94. It took *GrayScope* 6.97 s to localize the root cause of each gray failure on average.

To gain insights into the gray failure localization process of *GrayScope*, we utilize three cases from dataset $\boldsymbol{C}$ to illustrate the step-by-step workflow employed by *GrayScope*, as shown in Fig. 6.
(1) **CPU exhaustion.** When the Anomaly Detection module of *GrayScope* detected anomalies in the performance metric *sli_tps* of the GaussDB application, it triggered *GrayScope* to conduct root cause localization. *GrayScope* identified *cpu_user_msec* (the amount of time that the CPU has spent executing processes in user mode) as the primary culprit behind the gray failure and provides the possible propagation path of the gray failure. According to the results provided by *GrayScope*, this gray failure was attributed to an application process consuming a significant amount of CPU time (*cpu_user_msec* ↑), resulting in insufficient CPU resources for processing disk I/O, leading to disk operations being queued for processing (*disk_util* ↓), ultimately causing a decrease in the throughput of the GaussDB application (*sli_tps* ↓). With the root cause localization result, operators promptly investigated suspicious processes with high CPU utilization and quickly took measures to restore the performance of GaussDB.
(2) **Network latency.** When network instability led to increased response time (*tcp_link_srtt* ↑), the number of requests received by GaussDB from users within a unit of time decreased, resulting in reduced disk I/O demands (*disk_util* ↓), ultimately manifesting as degradation in GaussDB performance (*sli_tps* ↓). After receiving the gray failure ticket generated by *GrayScope*, operators inspected and configured network-related devices on the server OS, promptly mitigating this gray failure.
(3) **Disk IO high load.** The underlying logic involved suspicious processes heavily utilizing disk write bandwidth (*disk_wspeed* ↑), causing GaussDB to lack sufficient disk I/O resources for data write operations, resulting in performance degradation (*sli_tps* ↓). When operators received the gray failure ticket generated by *GrayScope*, they quickly identified and addressed suspicious processes with high disk write resource utilization.

Based on the above analysis, it is evident that the measures taken by *GrayScope* to provide gray failure paths offer greater convenience to operators and strengthen their acceptance of root causes.

5.2 Lessons Learned

Our experiments on gray failure cases from the industrial environment of *Huawei Cloud* reveal that the current industrial practice of gray failure root cause localization relies heavily on the intuition from operators' diagnosing experience. Therefore, this paper explores the significance of expert knowledge for accurate root cause localization. Furthermore, the anomaly scores of metrics, serving as preliminary evidence of potential root causes in the gray failures, are crucial for the effective operation of *GrayScope*. Based on the conducted experiments, we draw attention to two primary limitations of *GrayScope* as follows:
(1) *GrayScope* currently relies on expert knowledge when learning causal graphs, as operators typically expect failure localization solutions to be compatible with their existing knowledge. However, it may limit the flexibility and accuracy of *GrayScope*'s implementation. To resolve this issue, future iterations of *GrayScope* will explore mechanisms to enhance the autonomy of the causal learning process, which can learn and update causal graphs with minimal human intervention, thereby balancing the need for expert insight with the benefits of automated learning.
(2) In the current version of *GrayScope*, the input requires the anomaly scores of various metrics, which depend on the accuracy of the anomaly detection methods and the reliability of the anomaly score calculations. In the future, we plan to design a module to measure the degree of metric anomalies, eliminating the need to input anomaly scores into the method directly.

6 RELATED WORK

6.1 Causal Discovery

The PC algorithm [39] determines the presence or absence of edges (connections) between variables by employing statistical independence tests and conditional independence relationships and is particularly useful for inferring causal relationships from observational data. CauseInfer [4], MS-Rank [26], and AutoMap [28] apply PC algorithm to construct causal graphs based on performance metrics (*e.g.*, latency). ServiceRank [27] extracts the causal relationships between services. MicroCause [30] uses an improved PC algorithm to learn the causal relationship of each point of the time series. However, due to the high computational complexity, the PC algorithm requires a lot of computational resources and time, making it inappropriate for server OS with limited computational resources for gray failure localization. Moreover, the PC algorithm's high sensitivity to data distribution makes their results unstable.

The Granger causality test [10] has been used by many methods to analyze the causality of time series [31, 32, 40]. For instance, DyCause [32] performs the Granger causality test to get the degree of influence of the service on the front-end application and other services. However, these methods may learn inaccurate causality by overlooking the valuable knowledge of experts regarding causality, thereby degrading the performance of the gray failure localization.

Some other approaches employ novel techniques to learn causal relations. For example, REASON [42] proposes a hierarchical graph neural network-based causal discovery method to learn interdependent causation from multivariate time series. However, constructing and training hierarchical graph neural networks to model causal relationships require substantial computational resources, which is inappropriate for our scenario where only little computational resources can be used for gray failure localization in a server OS. FRL-MFPG [6] introduces a method (MFPG-FC) to map failure propagation, effectively handling dependencies in microservice architectures. However, it may face challenges in scenarios like server OS, where microservice-specific failure patterns do not apply.

6.2 Root Cause Localization

DFS-based methods [4, 22] and BFS-based methods [32, 36] traverse the graph more dependent on anomaly detection results. Both DFS and BFS can become computationally expensive and less efficient as the size of the metrics' causality graph increases. Moreover, these methods might localize multiple potential root causes without distinguishing which are more likely or impactful. Some works adopt random walk and its variants [6, 26–28, 30, 41, 42] or PageRank [25, 43, 48, 49] to rank and localize root causes. However, these works usually ignore the correlations between metrics and KPIs, making the root cause inference inaccurate. Some approaches [17, 19] link causal inference interventions to root cause localization, utilizing causal methods to discover the root cause. These approaches rely on the distribution of metric data to localize root causes, hence the accuracy of localization is susceptible to the data noise in server OS. Other approaches utilize machine learning techniques to automatically learn a microservices system's normal and various failure states from metrics, aiming to accomplish root cause localization tasks. For example, HRLHF [44] compares the temporal causal mechanism between normal and abnormal periods in microservices, aiding in anomaly attribution for each sub-component. TS-InvarNet [13] is based on the assumption of the stability of relationships between KPIs, aiming to mine and interpret state changes of invariants for root cause localization. Nevertheless, the performance of these approaches in gray failure localization for server OS is primarily hindered by their high computational cost that cannot be guaranteed in a server OS.

7 CONCLUSION

This paper investigates the root cause localization problem of gray failures in server OS. We introduce *GrayScope*, a methodology for localizing root causes by mining causal relationships between metrics and gray failure propagation patterns. Integrating expert knowledge with causal learning techniques ensures more reliable learning of metric causal graphs. The combination of partial correlation and anomaly degree enhances the accuracy of root cause inference. The recommendation of propagation paths enhances the interpretability of the results. We conducted extensive evaluation experiments based on the 1241 injected gray failures in *Huawei* to verify the effectiveness of *GrayScope*. Empirical results relying on the 135 gray failure cases in the industrial environment affirm the robustness and efficacy of *GrayScope* in achieving accurate and efficient root cause localization. We have made the source code publicly available for further research.

ACKNOWLEDGMENTS

This work is supported by the Advanced Research Project of China (No. 31511010501), and the National Natural Science Foundation of China (62272249, 62302244, 62072264).

REFERENCES

[1] Monica Bianchini, Marco Gori, and Franco Scarselli. 2005. Inside PageRank. *ACM Transactions on Internet Technology (TOIT)* 5, 1 (2005), 92–128. https://doi.org/10.1145/1052934.1052938

[2] Mattia Carletti, Chiara Masiero, Alessandro Beghi, and Gian Antonio Susto. 2019. Explainable Machine Learning in Industry 4.0: Evaluating Feature Importance in Anomaly Detection to Enable Root Cause Analysis. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, Los Alamitos, CA, 21–26. https://doi.org/10.1109/SMC.2019.8913901

[3] Chaosblade. 2024. Open Source Repository of Chaosblade. https://github.com/chaosblade-io/chaosblade

[4] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. 2014. CauseInfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, Los Alamitos, CA, 1887–1895. https://doi.org/10.1109/INFOCOM.2014.6848128

[5] Shanshan Chen, Xiaoxin Tang, Hongwei Wang, Han Zhao, and Minyi Guo. 2016. Towards Scalable and Reliable In-Memory Storage System: A Case Study with Redis. In *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, Los Alamitos, CA, 1660–1667. https://doi.org/10.1109/TrustCom.2016.0255

[6] Yuhua Chen, Dongqi Xu, Ningjiang Chen, and Xu Wu. 2023. FRL-MFPG: Propagation-aware fault root cause location for microservice intelligent operation and maintenance. *Information and Software Technology* 153 (2023), 107083. https://doi.org/10.1016/j.infsof.2022.107083

[7] Gala-anteater. 2024. Open Source Repository of Gala-anteater. https://gitee.com/openeuler/gala-anteater

[8] Gala-gopher. 2024. Open Source Repository of Gala-gopher. https://gitee.com/openeuler/gala-gopher

[9] Janos Gertler. 2002. Fault Detection and Diagnosis in Engineering Systems. *Control Engineering Practice* 9, 10 (2002), 1037–1038. https://doi.org/10.1201/9780203756126

[10] Clive WJ Granger. 1969. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica: journal of the Econometric Society* 37, 3 (1969), 424–438. https://doi.org/10.2307/1912791

[11] GrayScope. 2024. Open Source Repository of GrayScope. https://gitee.com/milohaha/grayscope

[12] Shiqi Hao, Yang Liu, Yu Wang, Yuan Wang, and Wenming Zhe. 2022. Three-Stage Root Cause Analysis for Logistics Time Efficiency via Explainable Machine Learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 2987–2996. https://doi.org/10.1145/3534678.3539024

[13] Zijun Hu, Pengfei Chen, Guangba Yu, Zilong He, and Xiaoyun Li. 2022. TS-InvarNet: Anomaly Detection and Localization based on Tempo-spatial KPI Invariants in Distributed Services. In *2022 IEEE International Conference on Web Services (ICWS)*. IEEE, Los Alamitos, CA, 109–119. https://doi.org/10.1109/ICWS55610.2022.00031

[14] Peng Huang, Chuanxiong Guo, Jacob R Lorch, Lidong Zhou, and Yingnong Dang. 2018. Capturing and Enhancing In Situ System Observability for Failure Detection. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 1–16.

[15] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. 2017. Gray Failure: The Achilles' Heel of Cloud-Scale Systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. ACM, New York, NY, USA, 150–155. https://doi.org/10.1145/3102980.3103005

[16] Ltd. Huawei Technologies Co. 2022. Introduction to Huawei Cloud Database GaussDB. In *Database Principles and Technologies–Based on Huawei GaussDB*. Springer, Berlin, 287–312. https://doi.org/10.1007/978-981-19-3032-4_8

[17] Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. 2022. Root Cause Analysis of Failures in Microservices through Causal Discovery. *Advances in Neural Information Processing Systems* 35 (2022), 31158–31170.

[18] M Frans Kaashoek, Dawson R Engler, Gregory R Ganger, and Deborah A Wallach. 1996. Server operating systems. In *Proceedings of the 7th workshop on ACM SIGOPS European workshop: Systems support for worldwide applications*. ACM, New York, NY, USA, 141–148. https://doi.org/10.1145/504450.504478

[19] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2022. Causal Inference-Based Root Cause Analysis for Online Service Systems with Intervention Recognition. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 3230–3240. https://doi.org/10.1145/3534678.3539041

[20] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, et al. 2022. Actionable and Interpretable Fault Localization for Recurring Failures in Online Service Systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, NY, 996–1008. https://doi.org/10.5281/zenodo.6955909

[21] Zhong Li, Yuxuan Zhu, and Matthijs Van Leeuwen. 2023. A Survey on Explainable Anomaly Detection. *ACM Transactions on Knowledge Discovery from Data* 18, 1 (2023), 1–54. https://doi.org/10.1145/3609333

[22] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. 2021. MicroHECL: High-Efficient Root Cause Localization in Large-Scale Microservice Systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, Los Alamitos, CA, 338–347. https://doi.org/10.1109/ICSE-SEIP52600.2021.00043

[23] Ping Liu, Yu Chen, Xiaohui Nie, Jing Zhu, Shenglin Zhang, Kaixin Sui, Ming Zhang, and Dan Pei. 2019. FluxRank: A Widely-Deployable Framework to Automatically Localizing Root Cause Machines for Software Service Failure Mitigation. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Los Alamitos, CA, 35–46. https://doi.org/10.1109/ISSRE.2019.00014

[24] Chang Lou, Peng Huang, and Scott Smith. 2020. Understanding, Detecting and Localizing Partial Failures in Large System Software. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Carlsbad, CA, 559–574.

[25] Xianglin Lu, Zhe Xie, Zeyan Li, Mingjie Li, Xiaohui Nie, Nengwen Zhao, Qingyang Yu, Shenglin Zhang, Kaixin Sui, Lin Zhu, et al. 2022. Generic and Robust Performance Diagnosis via Causal Inference for OLTP Database Systems. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, Los Alamitos, CA, 655–664. https://doi.org/10.1109/CCGrid54584.2022.00075

[26] Meng Ma, Weilan Lin, Disheng Pan, and Ping Wang. 2019. MS-Rank: Multi-Metric and Self-Adaptive Root Cause Diagnosis for Microservice Applications. In *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, Los Alamitos, CA, 60–67. https://doi.org/10.1109/ICWS.2019.00022

[27] Meng Ma, Weilan Lin, Disheng Pan, and Ping Wang. 2021. ServiceRank: Root Cause Identification of Anomaly in Large-Scale Microservice Architectures. *IEEE Transactions on Dependable and Secure Computing* 19, 5 (2021), 3087–3100. https://doi.org/10.1109/TDSC.2021.3083671

[28] Meng Ma, Jingmin Xu, Yuan Wang, Pengfei Chen, Zonghua Zhang, and Ping Wang. 2020. AutoMAP: Diagnose Your Microservice-based Web Applications Automatically. In *Proceedings of The Web Conference 2020*. ACM, New York, NY, USA, 246–258. https://doi.org/10.1145/3366423.3380111

[29] Luciano Manelli, Giulio Zambon, Luciano Manelli, and Giulio Zambon. 2020. Introducing JSP and Tomcat. *Beginning Jakarta EE Web Development: Using JSP, JSF, MySQL, and Apache Tomcat for Building Java Web Applications* (2020), 1–53.

[30] Yuan Meng, Shenglin Zhang, Yongqian Sun, Ruru Zhang, Zhilong Hu, Yiyin Zhang, Chenyang Jia, Zhaogang Wang, and Dan Pei. 2020. Localizing Failure Root Causes in a Microservice through Causality Inference. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, Los Alamitos, CA, 1–10. https://doi.org/10.1109/IWQoS49365.2020.9213058

[31] Meike Nauta, Doina Bucur, and Christin Seifert. 2019. Causal Discovery with Attention-Based Convolutional Neural Networks. *Machine Learning and Knowledge Extraction* 1, 1 (2019), 19. https://doi.org/10.3390/make1010019

[32] Yicheng Pan, Meng Ma, Xinrui Jiang, and Ping Wang. 2021. Faster, deeper, easier: crowdsourcing diagnosis of microservice kernel failure from user space. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, New York, NY, 646–657. https://doi.org/10.1145/3460319.3464805

[33] Judea Pearl et al. 2000. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress* 19, 2 (2000), 3.

[34] Karl Pearson. 1905. The Problem of the Random Walk. *Nature* 72, 1865 (1905), 294–294. https://doi.org/10.1038/072342a0

[35] Prometheus. 2024. Open Source Repository of Prometheus. https://github.com/prometheus/prometheus

[36] Juan Qiu, Qingfeng Du, Kanglin Yin, Shuang-Li Zhang, and Chongshu Qian. 2020. A Causality Mining and Knowledge Graph Based Method of Root Cause Diagnosis for Performance Anomaly in Cloud Applications. *Applied Sciences* 10, 6 (2020), 2166. https://doi.org/10.3390/app10062166

[37] Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. 2019. ϵ-diagnosis: Unsupervised and Real-time Diagnosis of Small- window Long-tail Latency in Large-scale Microservice Platforms. In *The World Wide Web Conference*. ACM, New York, NY, USA, 3215–3222. https://doi.org/10.1145/3308558.3313653

[38] Jacopo Soldani and Antonio Brogi. 2022. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *ACM Computing Surveys (CSUR)* 55, 3 (2022), 1–39. https://doi.org/10.1145/3501297

[39] Peter Spirtes and Clark Glymour. 1991. An Algorithm for Fast Recovery of Sparse Causal Graphs. *Social science computer review* 9, 1 (1991), 62–72. https://doi.org/10.1177/089443939100900106

[40] Alex Tank, Ian Covert, Nicholas Foti, Ali Shojaie, and Emily B Fox. 2021. Neural Granger Causality. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2021), 4267–4279. https://doi.org/10.1109/TPAMI.2021.3065601

[41] Dongjie Wang, Zhengzhang Chen, Yanjie Fu, Yanchi Liu, and Haifeng Chen. 2023. Incremental Causal Graph Learning for Online Root Cause Analysis. In

Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. ACM, New York, NY, USA, 2269–2278. https://doi.org/10.1145/3580305.3599392

[42] Dongjie Wang, Zhengzhang Chen, Jingchao Ni, Liang Tong, Zheng Wang, Yanjie Fu, and Haifeng Chen. 2023. Interdependent Causal Networks for Root Cause Localization. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 5051–5060. https://doi.org/10.1145/3580305.3599849

[43] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selcuk Kopru, and Tao Xie. 2021. Groot: An Event-graph-based Approach for Root Cause Analysis in Industrial Settings. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Los Alamitos, CA, 419–429. https://doi.org/10.1109/ASE51524.2021.9678708

[44] Lu Wang, Chaoyun Zhang, Ruomeng Ding, Yong Xu, Qihang Chen, Wentao Zou, Qingjun Chen, Meng Zhang, Xuedong Gao, Hao Fan, et al. 2023. Root Cause Analysis for Microservice Systems via Hierarchical Reinforcement Learning from Human Feedback. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 5116–5125. https://doi.org/10.1145/3580305.3599934

[45] Gary White, Jaroslaw Diuwe, Erika Fonseca, and Owen O'Brien. 2021. MMRCA: MultiModal Root Cause Analysis. In *International Conference on Service-Oriented Computing*. Springer, Springer-Verlag, Berlin, 177–189. https://doi.org/10.1007/978-3-031-14135-5_14

[46] Canhua Wu, Nengwen Zhao, Lixin Wang, Xiaoqin Yang, Shining Li, Ming Zhang, Xing Jin, Xidao Wen, Xiaohui Nie, Wenchi Zhang, et al. 2021. Identifying Root-Cause Metrics for Incident Diagnosis in Online Service Systems. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Los Alamitos, CA, 91–102. https://doi.org/10.1109/ISSRE52982.2021.00022

[47] Han Wu, Zhihao Shang, and Katinka Wolter. 2020. Learning to Reliably Deliver Streaming Data with Apache Kafka. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Los Alamitos, CA, 564–571. https://doi.org/10.1109/DSN48063.2020.00068

[48] Li Wu, Johan Tordsson, Jasmin Bogatinovski, Erik Elmroth, and Odej Kao. 2021. MicroDiag: Fine-grained Performance Diagnosis for Microservice Systems. In *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*. IEEE, Los Alamitos, CA, 31–36. https://doi.org/10.1109/CloudIntelligence52565.2021.00015

[49] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. 2020. MicroRCA: Root Cause Localization of Performance Issues in Microservices. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, Los Alamitos, CA, 1–9. https://doi.org/10.1109/NOMS47738.2020.9110353

[50] 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 *Proceedings of the Web Conference 2021*. ACM, New York, NY, USA, 3087–3098. https://doi.org/10.1145/3442381.3449905

[51] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, et al. 2020. Understanding and handling alert storm for online service systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. ACM, New York, NY, USA, 162–171. https://doi.org/10.1145/3377813.3381363

[52] Minghui Zhou, Xinwei Hu, and Wei Xiong. 2022. openEuler: Advancing a Hardware and Software Application Ecosystem. *IEEE Software* 39, 2 (2022), 101–105. https://doi.org/10.1109/MS.2021.3132138

Received 2024-02-08; accepted 2024-04-18