Generic and Robust Performance Diagnosis via Causal Inference for OLTP Database Systems

Xianglin Lu*, Zhe Xie†, Zeyan Li*, Mingjie Li*, Xiaohui Nie‡, Nengwen Zhao*, Qingyang Yu*,
Shenglin Zhang§**, Kaixin Sui‡, Lin Zhu¶, Dan Pei*
*Tsinghua University †Shanghai Jiao Tong University ‡BizSeer §Nankai University ¶China Mobile Research
** Haihe Laboratory of Information Technology Application Innovation (HL-IT)

2022 22nd International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 978-1-6654-9956-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/CCGRID54584.2022.00075

***Abstract*—Online transaction processing (OLTP) database systems provide an effective solution to data support for online applications with high concurrency and low latency. An interruption or performance degradation of OLTP database systems may impact the availability of services and bring substantial economic loss. Thus, diagnosing the issue timely and mitigating it rapidly are essential for database administrators (DBAs). However, performance diagnosis for database systems is challenging due to numerous abnormal metrics, complex failure propagation, and high-performance requirements. Existing works relying on anomaly detection or causal graph construction cannot handle all these challenges simultaneously. In this paper, we propose an unsupervised learning-based method, *CauseRank*, to perform root cause localization with superior efficiency, high accuracy, and good interpretability. Two key techniques in *CauseRank* are a novel causal discovery algorithm named Group-based Greedy Equivalent Search (G-GES) incorporated with domain knowledge which treats metric groups as nodes to capture failure propagation and a simple yet effective ranking method named Causal Oriented Personalized PageRank (COPP). Extensive experiments on 97 real-world failure cases collected from a large-scale Oracle database demonstrate the effectiveness of *CauseRank*, achieving 82.5% top-3 accuracy and 93.8% top-5 accuracy and outperforming baseline approaches. The core idea and framework of *CauseRank* are generic and can be applied to other large-scale system components.**

***Index Terms*—OLTP database systems, performance diagnosis, causal inference**

I. Introduction

Online Transaction Processing (OLTP) database systems are well known for the characteristics of high concurrency and low latency, which are widely used in many online applications, including online banking, shopping, and instant messaging. However, since the system allows a large number of concurrent transactions to modify data, it may malfunction due to internal or external reasons [1]. An interruption or performance degradation of OLTP database systems may cause service unavailability, severe economic loss, and reputation damage to enterprises [2]. For example, the downtime for over 12 hours of Salesforce's systems due to a database failure costs as high as 20 million dollars[1]. Therefore, it is crucial for DBAs (Database Administrators) to continuously monitor the performance of OLTP database systems and conduct failure diagnosis and recovery timely.

¶Lin Zhu is the corresponding author.

[1]https://content.dsp.co.uk/how-database-failure-can-impact-your-business

Most OLTP database systems automatically collect detailed statistics to monitor system performance through OLTP Database Management Systems (DBMSs) such as Oracle, DB2, MySql, and SQL Server [3]. For instance, in Oracle [2], Average Active Sessions (AAS) is a critical metric to characterize the performance of the database system, and it can be used to determine whether problems exist [4]. When such metrics behave abnormally, it means the service quality of the database system is affected. Seasoned DBAs usually localize the root cause manually based on years of accumulated expert experience to restore the system to normal operation. However, the manpower of DBAs is limited. In large-scale OLTP database systems, DBAs have to manually analyze thousands of metrics [1], which is labor-intensive and time-consuming. Therefore, automated root cause localization approaches are of great necessity to assist DBAs in rapid failure diagnosis and recovery.

In the literature, several works adopt supervised methods to localize the root causes [5], [6]. For example, MEPFL [5] only focuses on three common types of failures and uses supervised machine learning techniques to predict the specific types. Although these supervised methods are accurate and efficient, they need sufficient labeled data, which is quite difficult to obtain in practice. Some works propose to localize the root cause through learning the patterns of historical failures [3], [7], [8]. For example, iSQUAD [3] matches a new failure to historical failure clusters for root cause identification. These methods appear to be label-independent. However, in order to further diagnose the failure, the annotations or solutions of historical failures are still required after matching. Moreover, these methods fail to handle new types of failures (*i.e.*, failures that have never occurred in history) since the patterns of new-type failures cannot match those of historical failures.

Due to the above limitations, localization schemes without the dependence on historical failures are widely proposed. FluxRank [9] and ϵ-Diagnosis [10] localize the root cause based on anomaly degrees. However, these methods can be biased because a failure can introduce a large number of metrics with similar abnormal behaviors (thus with similar anomaly degrees) due to complex dependencies and failure propagation. Some prior works [11]–[15] take both anomaly degrees and causal analysis into consideration. Despite the fact that this kind of methods can handle new-type failures and provide

978-1-6654-9956-9/22/$31.00 ©2022 IEEE
DOI 10.1109/CCGrid54584.2022.00075

interpretations, they cannot ensure high accuracy because of ignoring domain knowledge. Moreover, these methods also suffer from low computational efficiency since considerable anomaly metrics bring enormous causal graph construction complexity. In a database system, there are always multiple metrics describing the same module of the system (*e.g.*, CPU, Memory, and Disk), whose anomalies imply the same kind of problem. For example, both the excessive increase of CPU utilization and that of CPU load average indicate that the system is overloaded. However, existing causal discovery algorithms scarcely consider the meanings of these metrics. They usually directly construct a causal graph with each node containing one single metric, leading to fuzzy relationships among metrics and a complex causal graph with a large number of edges, which negatively impacts localization and interpretability. In summary, designing an automated, unsupervised, accurate, and efficient root cause localization approach to tackle the above limitations of existing works is in urgent need.

In this work, designing such an approach faces the following significant challenges.

- **Numerous abnormal metrics.** The sharp increase in the number of abnormal metrics that behave similarly due to complex dependencies and failure propagation increases the hindrance for root cause localization.
- **Complex failure propagation.** Failure propagation follows a certain path. Finding the correct propagation path is beneficial to localizing the root cause and providing interpretations for DBAs. However, it is difficult to model the failure propagation accurately without the support of domain knowledge.
- **High performance requirements.** The failure localization method is designed to assist DBAs in faster troubleshooting. Thus, it is required to be accurate and efficient.

To address the above challenges together, in this paper, we propose a novel approach named *CauseRank*. *CauseRank* performs unsupervised learning on the time series data for a time period close to the occurrence of each failure. At the very beginning, *CauseRank* divides the metrics in the system into metric groups according to their belonging modules since localizing the root cause metric group is enough for determining subsequent mitigation actions. In this way, the first challenge is addressed. For online diagnosis, *CauseRank* utilizes metric group analysis as its building block. First, *CauseRank* generates candidate metric groups related to the failure by measuring the fluctuation of each metric. Second, *CauseRank* applies a novel causal discovery algorithm, Group-based Greedy Equivalent Search (G-GES), to build a temporary causal graph with metric groups. It is also incorporated with domain knowledge to obtain a more accurate failure propagation path, addressing the second challenge. *CauseRank* also adopts a penalty term during causal discovery to dynamically preserve the relatively important edges, which improves efficiency and tackles the third challenge. Third, we propose a simple yet effective algorithm named Causal Oriented Personalized PageRank (COPP) for metric group ranking, which further improves the performance of *CauseRank*.

To sum up, our contributions can be concluded as follows.

- We propose an unsupervised approach named *CauseRank* to localize the root cause via causal inference with high accuracy and efficiency for OLTP database systems, which can also provide interpretations to assist DBAs in rapid failure diagnosis and mitigation.
- In *CauseRank*, we propose a novel causal discovery algorithm, G-GES, which treats metric groups as nodes to build a causal graph and effectively captures failure propagation. We also adopt a potent strategy to integrate domain knowledge into G-GES to mitigate false inferences and enhance accuracy. Moreover, we improve the ranking algorithm and propose COPP to rank the metric groups.
- We extensively validate *CauseRank* on an Oracle dataset collected from a large commercial bank, which contains 97 real-world failures. Results show that *CauseRank* ranks the root causes at top-5 for 93.8%, top-3 for 82.5% of all the failures, with an average localization time of 12.58 seconds per failure. The mean average rank of the root causes using *CauseRank* is 2.13, outperforming the baseline methods by at least 33.6%. We also perform several additional experiments for ablation study to demonstrate the contributions of the key techniques in *CauseRank*.

II. Background and Overview

A. OLTP Database Systems

Online Transaction Processing (OLTP) database systems support concurrent transaction execution under the compliance of ACID principle [16] to maintain a high level of data integrity, which is widely used in online applications to achieve data storage, access, and update purposes. The DBMS is one of the core components in the database system, which is a layer between the users and the operating system to provide data definition and manipulation, effective and convenient interfaces, as well as the organization and management [17], [18], including concurrency control, database transaction processing, *etc*. It collects massive volumes of detailed statistics and logs to reveal system service quality and internal status [3]. Moreover, the DBMS usually integrates monitoring software to consolidate the statistics and present them to DBAs in a graphical interface, such as the Oracle database QoS (Quality of Service) management system. DBAs can configure monitoring items and alerts through such software, among which time-series metrics are the most concerned item in system maintenance. When the value of a certain metric reaches the threshold configured by DBAs, an alert will be generated. DBAs monitor the service quality through the DBMS to ensure the stability and high availability of the database system.

B. Problem Statement

When the OLTP database system cannot continue to provide service or the operation of the system affects user experience, we consider a failure occurs, which needs to be checked as soon as possible to restore high availability. It is time-consuming and error-prone for DBAs to investigate such a

TABLE I
METRIC GROUPS AND EXAMPLE METRICS

Metric group	Metrics
AAS (1)	Average Active Session (AAS)
ADR (3)	Automatic Diagnostic Repository (ADR) block file write, ADR block file read, ADR file lock
CPU (5)	CPUEntlUtil, CPUCpuUtil, CPUT, CPUWio, CPU
DBFile (7)	db file sequential read average wait time, db file parallel read, db file parallel write, db file sequential read, db file single write, db file scattered read, db file async IO submit
Disk (4)	disk file mirror read, disk file mirror/media repair write, disk file operations IO, disk busy percentage
Execution (4)	#executions, #logins, #connections, connection utilization
LogicRead (2)	logical read per execution, #logical read
LogFile (10)	log file single write, log file sync, log archive I/O, log file sequential read, log file switch completion, log file sync average wait time, log file parallel write, log file parallel write average wait time, log writer wait for redo copy, log file switch (checkpoint incomplete)
Memory (5)	memory utilization, udp bufferoverflows, PGA allocated, UpAvailabelMemPct, UpMEMComputePct
Transaction (2)	transaction count, #transactions per second

large number of metrics for localization manually. Therefore, our goal is to localize the root causes automatically.

Metrics. We divide the metrics of database systems into two categories: key metrics and infrastructure metrics. In database systems, DBAs always pay close attention to the alerts of key metrics that indicate the overall availability of the systems, *e.g.*, the AAS metric in Oracle. And others recording the status of each module of the system are called infrastructure metrics. These time-series metrics are what we focus on.

Metric groups. We divide the above metrics into metric groups for the following reasons. First, the number of metrics is huge in database systems, which could affect the performance of failure propagation modeling due to a large number of nodes and edges. Besides, there are multiple different metrics describing the status of the same module (*e.g.*, "CPU utilization" and "CPU idle" actually describe the characteristics of CPU from different perspectives). These metrics can be grouped to reduce the complexity of causal graphs. Second, metric anomalies within the same group indicate failures of the same module. Usually, DBAs perform the same failure mitigation operation when they discover that root cause metrics belong to the same metric group. Therefore, all key metrics are grouped into one key metric group since these metrics reflect the overall system availability from different perspectives. Moreover, the infrastructure metrics are categorized into different metric groups according to their belonging modules by DBAs. Due to the limitation of space, we only list ten example metric groups used in the experiment in Table I for intuitive illustrations, where the number of metrics in each group is recorded in (). Note that the manual metric grouping is not the main contribution of our work. DBAs can refine the grouping method in line with actual operating environments. Later, we will introduce how we utilize the metric groups to localize the root cause in Sec. III.

Problem description. With the above definitions, the objective of this work can be described as follows. During the operation of the OLTP database system, key metrics are continuously monitored. A failure occurs when a key metric anomaly is detected, implying that there may be a problem with the system availability. Then, *CauseRank* utilizes the time series data of key metrics and infrastructure metrics for a time period close to the failure occurrence to rank potential top-N metric groups as the most likely root causes.

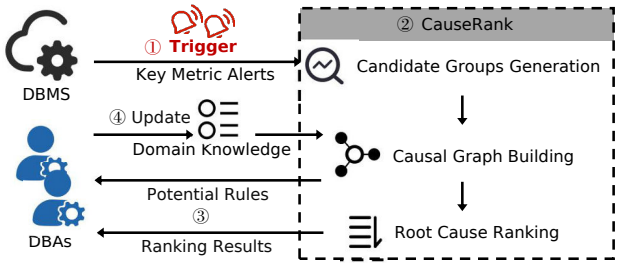


Fig. 1. System workflow of *CauseRank*.

C. Design Overview

The workflow of *CauseRank* is shown in Fig. 1, where the stages in the dashed box are what *CauseRank* focuses on. Suppose that an alert of any key metric is observed in the database system, which indicates a failure happens. *CauseRank* immediately starts to localize the root cause.

First, *CauseRank* generates candidate metric groups from all metric groups by measuring the fluctuation of each metric and then calculates the anomaly score for each metric group. Second, *CauseRank* applies a novel algorithm, G-GES, to temporarily construct a causal graph based on these metric groups accurately and efficiently, which is enhanced by existing domain knowledge. We construct the causal graph per failure to capture the propagation pattern of only the current failure and obtain a more accurate causal graph. Potential rules discovered by G-GES can be leveraged to enrich domain knowledge after being confirmed by DBAs. Third, COPP is applied to rank the candidate infrastructure metric groups on the constructed causal graph. The ranking scores obtained through COPP are integrated with the anomaly scores to obtain each candidate metric group's final root cause score. All possible propagation paths from each potential root cause metric group to the key metric group are delivered to DBAs for interpretability faster troubleshooting.

III. METHODOLOGY

In this section, we present the details of *CauseRank*, which utilizes metric group analysis as its building block. In *CauseRank*, we first generate candidate metric groups to initialize nodes (Sec. III-A), and then we develop a novel graph building algorithm (G-GES in Sec. III-B) as well as the corresponding ranking technique (COPP in Sec. III-C) to address the challenges mentioned in Sec. I.

A. Candidate Groups Generation

CauseRank is supposed to be triggered whenever a failure arises. Denote the earliest time that the failure is observed

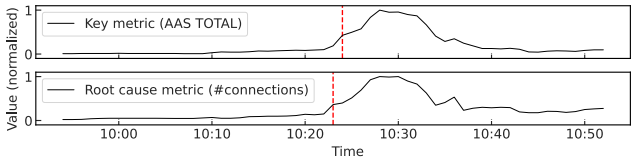


Fig. 2. An example of a failure with a key metric and the root cause infrastructure metric.

as t_{alert}. To obtain deeper insights, we need to filter out suspicious infrastructure metric groups first.

For a certain infrastructure metric group (denoted as X), *CauseRank* measures the fluctuation of each metric x in X compared with its normal state to evaluate how suspicious X is. An effective and widely used method (known as *n-sigma*) is to model the normal state of x with its mean μ_x and standard deviation σ_x. *CauseRank* estimates the anomaly degree of x for each data point $x^{(t)}$ around t_{alert} with $s_x = \max_t |x^{(t)} - \mu_x|/\sigma_x$ based on Z-Score. With s_x, the final anomaly score of x (denoted as α_x) is defined by (1). α_x considers an infrastructure metric that satisfies $s_x < k$ as normal, where k is determined by the fluctuation level of the system during normal operation and DBAs' tolerance for anomalies. In this work, we take $k = 1$ to avoid missing root cause metrics with slight fluctuations. Moreover, α_x applies logarithmic conversion to s_x as it limits a high s_x getting higher to avoid extreme situations.

$$\alpha_x = \begin{cases} \log(1 + s_x), & s_x \geq k \\ 0, & s_x < k \end{cases} \tag{1}$$

We use the data points in the time period $[t_{alert} - \ell_p - \ell_{train}, t_{alert} - \ell_p]$ to learn μ_x and σ_x for each infrastructure metric x, where ℓ_{train} denotes the training data length, which is required to be long enough to model historical normal state more accurately. We shift the period with ℓ_p as the system may already be abnormal before the failure is observed at t_{alert}. For example, Fig. 2 shows the time delay between the occurrence of an actual failure observed in an Oracle database (indicated by the root cause metric) and the impact on the key metric, which are marked with two red lines. For the test data, we use the data points in the time period $[t_{alert} - \ell_p, t_{alert} + \ell_{test}]$, where ℓ_{test} denotes the test data length, which is decided by the real-time requirement of the system for root cause localization. However, ℓ_{test} should be large enough to prevent the localization result from being affected by noise.

Finally, for each infrastructure metric group, the metrics with $\alpha_x > 0$ are retained, and the others are not considered in this failure. The largest anomaly score of the metrics in each group is taken as the corresponding group score α_X. All the groups with anomaly metrics are considered candidate metric groups (including the key metric group), which will be used as nodes to construct the causal graph in the next stage.

B. Causal Graph Building

To address the challenge of failure propagation modeling to provide interpretability, *CauseRank* explicitly models the propagation path among candidate metric groups generated in Sec. III-A as causal relations. In *CauseRank*, we propose an efficient causal discovery algorithm called G-GES (Group-based Greedy Equivalent Search) with novelty. G-GES takes metric groups as vertices to infer the causality among them, effectively reducing the complexity of the graph caused by the excessive number of metrics and providing a more concise and clear causal graph whose edges point from cause to result. To improve the accuracy of the causal graph, we also integrate domain knowledge in the building process, which plays an essential role in improving the effectiveness of root cause localization as described in Sec. IV-B. The metrics of a normal system fluctuate relatively smoothly, and the causal relations reflected in different failures are not always the same, which may conceal some causal relations of the current failure. Therefore, *CauseRank* constructs a temporary causal graph every time a failure occurs to model a more accurate failure propagation path.

1) Greedy Equivalent Search (GES): GES [19] is a classic score-based causal discovery algorithm whose principle is to maximize the overall score of the graph with maximum likelihood estimation (MLE) through continuous iterations to build the causal graph. Equation (2) shows the general definition of scoring criterion S using the relative log posterior [19], where $\mathcal{G}$ is a hypothesis causal graph and $\mathcal{D}$ is the observed data.

$$S(\mathcal{G}, \mathcal{D}) = \log p(\mathcal{G}) + \log p(\mathcal{D}|\mathcal{G}) \tag{2}$$

According to the structure of a directed acyclic graph $\mathcal{G}$, the metric datum $\mathbf{x}^{(t)}$ at time t can be factorized as (3), where $pa_{\mathcal{G}}(x)$ is the set of parent nodes of x in $\mathcal{G}$.

$$p(\mathbf{x} = \mathbf{x}^{(t)}|\mathcal{G}) = \prod_{x \in \mathbf{x}} p(x = x^{(t)}|pa_{\mathcal{G}}(x) = pa_{\mathcal{G}}^{(t)}(x)) \tag{3}$$

Hence, the relative log posterior is decomposable as shown in (4), where $s(x, pa_{\mathcal{G}}(x))$ is short for $s(x, pa_{\mathcal{G}}(x)|\mathcal{D})$, called *local score*.

$$\begin{aligned} & S(\mathcal{G}, \mathcal{D}) \\ = \; & \log p(\mathcal{G}) + \log \prod_t p(\mathbf{x} = \mathbf{x}^{(t)}|\mathcal{G}) \\ = \; & \log p(\mathcal{G}) + \sum_{x \in \mathbf{x}} \log \prod_t p(x = x^{(t)}|pa_{\mathcal{G}}(x) = pa_{\mathcal{G}}^{(t)}(x)) \\ = \; & \log p(\mathcal{G}) + \sum_{x \in \mathbf{x}} s(x, pa_{\mathcal{G}}(x)) \end{aligned} \tag{4}$$

GES estimates the causal graph G through maximizing the score $S(\mathcal{G}, \mathcal{D})$ with two steps called Greedy Forward Search (GFS) and Greedy Backward Search (GBS), respectively. GES starts with an empty graph. In GFS (GBS), it iteratively adds (deletes) every possible edge to (from) the graph, until $S(\mathcal{G}, \mathcal{D})$ will not increase any more.

2) Group-based Greedy Equivalent Search (G-GES): G-GES is inspired by GES. The improvements are mainly two folds: 1) G-GES takes metric groups as the nodes in the causal graph instead of a single metric; 2) G-GES modifies the score calculation mechanism of GES. Overall, G-GES is composed

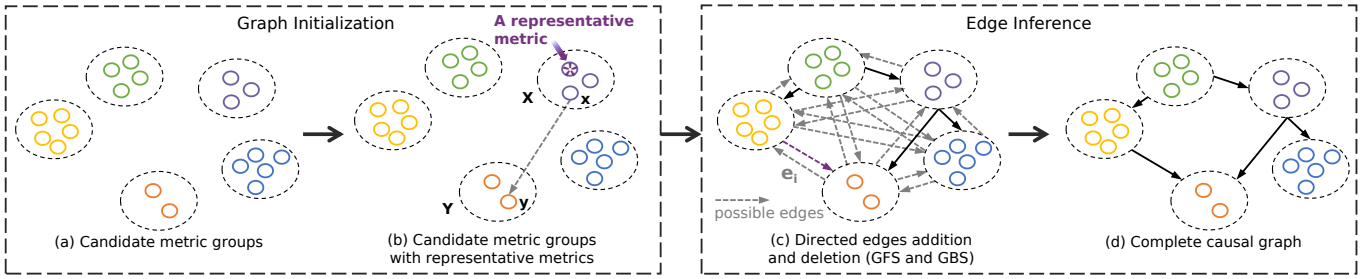


Fig. 3. The structure of G-GES.

of two phases: graph initialization and edge inference as shown in Fig. 3, where each small circle is a metric, whose color represents its metric group.

a) Graph initialization: In this phase, the *representative metrics* are chosen for each metric group pair. As shown in Fig. 3a, dashed larger circles represent metric groups, which are treated as nodes in the causal graph. We infer the causal graph with the metrics in the time period $[t_{alert} - \ell_{graph}, t_{alert} + \ell_{test}]$, where ℓ_{graph} denotes the historical reference length. We select *representative metrics* for each metric group pair to accelerate the edge inference phase. For each pairwise groups (*i.e.*, nodes) X and Y obtained from the permutation of all the metric groups, we choose a *representative metric* from X for Y, denoted as M_{XY} (as presented in Fig. 3b with a star in a circle), which is the metric in group X that best reflects the causality with group Y. In order to calculate M_{XY}, we use GFS (Greedy Forward Search) to calculate the improved score when only edge e_{xy} exists compared with the empty graph, where e_{xy} is an edge from metric x in group X to metric y in group Y. The calculation of M_{XY} can be expressed with the *local score* s as shown in (5), and it will later be used to infer the edges when the graph presents different states.

$$M_{XY} = \arg\max_{x \in X} \left\{ \max_{y \in Y} [s(y, \{x\}) - s(y, \emptyset)] \right\} \tag{5}$$

b) Edge inference: In this phase, the causality of the metric groups is mined to obtain directed edges. G-GES also contains two successive steps: GFS and GBS. The difference is that G-GES adjusts the score calculation mechanism in these two steps to tackle grouped metrics in root cause localization scenario. We denote the adjusted scoring criterion and *local score* as S_G and s_G respectively. At first, there is an empty graph denoted as $\mathcal{G}_0$. In the GFS (GBS) step, edges that improve the total score of the graph the most are continuously added to (deleted from) the graph through multiple iterations. In the i-th iteration of GFS (GBS), the non-existent edges (represented by the gray dashed arrows in Fig. 3c, while in GBS, there need the existent edges represented by black solid arrows) in the current graph $\mathcal{G}_i(V_i, E_i)$ are possible edges that can be added into (deleted from) the graph. When a possible directed edge e_{XY} from X to Y is added (deleted) in $\mathcal{G}_i$, we obtain a temporary graph $\mathcal{G}'_i(V'_i, E'_i)$. Denote the change of the corresponding relative log posterior as $\Delta S_G(\mathcal{G}_i, \mathcal{D}, e_{XY})$. Under the assumption of a uniform prior of the causal graph [19], based on (4), $\Delta S_G(\mathcal{G}_i, \mathcal{D}, e_{XY})$ is the difference between the related *local score* as shown in (6), where $Pa_{\mathcal{G}'_i}(Y) = Pa_{\mathcal{G}_i}(Y) \cup \{X\}$ (in GBS, $Pa_{\mathcal{G}'_i}(Y) = Pa_{\mathcal{G}_i}(Y) \setminus \{X\}$).

$$\begin{aligned} \Delta S_G(\mathcal{G}_i, \mathcal{D}, e_{XY}) =& S_G(\mathcal{G}'_i, \mathcal{D}) - S_G(\mathcal{G}_i, \mathcal{D}) \\ =& s_G(Y, Pa_{\mathcal{G}'_i}(Y)) - s_G(Y, Pa_{\mathcal{G}_i}(Y)) \end{aligned} \tag{6}$$

And $s_G(Y, Pa_{\mathcal{G}_i}(Y))$ of graph $\mathcal{G}_i$ can be calculated by (7), where $\{M_{IY} | I \in Pa(Y)\}$ indicates the *representative metrics* of Y's parent to node Y in $\mathcal{G}_i$ and $Pa(Y)$ is short for $Pa_{\mathcal{G}_i}(Y)$.

$$s_G(Y, Pa(Y)) = \max_{y \in Y} s(y, \{M_{IY} | I \in Pa(Y)\}) \tag{7}$$

We adopt $\ell 0$-penalized Gaussian maximum likelihood to estimate s as shown in (8), which is implemented with the `ges` package[2]. N denotes the number of data points, while σ is the variance of residual by doing a linear regression of y on $\{M_{IY} | I \in Pa(Y)\}$. The penalty term $-\omega(|Pa(Y)|+1)$ limits the length of $Pa(Y)$ and preserves the most critical edges as well as reduces the false positives so as to obtain a sparser graph and improve efficiency.

$$\hat{s}(y, \{M_{IY} | I \in Pa(Y)\}) = -\frac{N}{2}(1+\log(\sigma)) - \omega(|Pa(Y)|+1) \tag{8}$$

Then, as shown in (9), the edge e_i that improves the score the most will be finally added to $\mathcal{G}_i$ in GFS (deleted from $\mathcal{G}_i$ in GBS) in the i-th iteration to obtain $\mathcal{G}_{i+1}$ until the score will not increase anymore.

$$e_i = \arg\max_{e_{XY}} \Delta S_G(\mathcal{G}_i, \mathcal{D}, e_{XY}) \tag{9}$$

Through multiple iterations of GFS and GBS respectively, we can obtain the causal graph $\mathcal{G}(V, E)$ constructed by G-GES as demonstrated in Fig. 3d.

3) Enhancement with Domain Knowledge: However, when building the causal graph, G-GES only considers the value of each metric, but nothing about its actual meaning or its relations with other metrics. Considering that in general situations, we usually have some external domain knowledge about the system provided by DBAs, we can obtain an expert causal graph with the expert rules inspired by domain knowledge. Each of the expert rules is a confirmed causal relation between two metric groups or two metrics. And this graph can be used to direct G-GES to build the complete causal graph.

Nevertheless, we still need to run G-GES after getting an expert causal graph since the expert rules are incomplete,

[2]https://github.com/juangamella/ges

especially in unknown failure scenarios. The quantity and quality of the rules mainly depend on the DBAs' empirical knowledge and how much efforts DBAs put in. Moreover, even if we have a complete expert causal graph, there is no guarantee that all the expert rules among the metrics in the graph always exist in any scenario, resulting in incorrect failure propagation paths and inaccurate root cause localization.

Suppose that the expert causal graph generated on the basis of domain knowledge is $G_D(V_D, E_D)$. We update the calculation of $\Delta S_G(\mathcal{G}_i, \mathcal{D}, e_{XY})$ in the GFS step of G-GES with (10), where constant β satisfies $\beta \geq 1$, which represents the augmented weight of domain knowledge.

$$\Delta S_G^*(\mathcal{G}_i, \mathcal{D}, e_{XY}) = \begin{cases} \Delta S_G(\mathcal{G}_i, \mathcal{D}, e_{XY}) \cdot \beta, & e_{XY} \in E_D \\ \Delta S_G(\mathcal{G}_i, \mathcal{D}, e_{XY}), & e_{XY} \notin E_D \end{cases} \tag{10}$$

Such scheme ensures that during the computation of *CauseRank*, edges in G_D are more likely to be added to the final causal graph promptly, which helps us obtain a more accurate causal graph. After the construction, we collect the edges in the inferred causal graph but not included in the expert causal graph and record them as potential rules, which are recommended to DBAs for further confirmation. Those confirmed rules will be added to the expert rules, achieving an active update of the expert causal graph.

C. Root Cause Ranking

Once we build a causal graph, another problem is how we rank the nodes in the causal graph to figure out the root cause. Existing works [12], [14], [20] mostly rely on PageRank. They set up the weight of each edge by calculating the similarity between the metrics of two connected nodes or the metric of end node of the edge and the anomaly key metrics. However, this may result in false positives since the similarity is symmetric, which cannot reveal the complete causality between every two metrics. Moreover, it cannot be directly applied to the scenario where metric groups are considered nodes.

We propose Causal Oriented Personalized PageRank (COPP) based on Personalized PageRank to solve these issues. Compared with naive PageRank, Personalized PageRank [21] optimizes the probability of each moving step by adding the personalization for each node. In the failure localization scenario, this kind of personalization can be effectively used to activate each failure's preference for different infrastructure metric groups, which is more practical. COPP contains two phases, namely graph setup and node ranking.

1) Graph Setup: The causal graph we obtained from G-GES incorporated with domain knowledge cannot be directly used for root cause ranking since each edge of it points from cause to result without weight. Our solution is to convert the causal graph to a relational graph.

First, we calculate the weight for each edge e_{XY} in the causal graph $\mathcal{G}(V, E)$ obtained by Sec. III-B through (11), where $\mathcal{G}'(V', E')$ is the graph without edge e_{XY}.

$$\textit{Weight}(e_{XY}) = -(S_G(\mathcal{G}', \mathcal{D}) - S_G(\mathcal{G}, \mathcal{D})) \tag{11}$$

The weights are all positive since the edges that have negative contributions to the entire graph are deleted in the GBS step. The weight calculation step is similar to GBS in G-GES. The difference is that the complete causal graph $\mathcal{G}$ remains unchanged in the weight calculation step, while it may change in each iteration of GBS. Then, we flip the direction of each edge in the causal graph whose edges are from cause to result and obtain the relational graph with a weight matrix A, where A_{IJ} represents the weight of the edge from I to J.

By definition, the Personalized PageRank algorithm is enforced to move to another node even if the causal relations between the current node and the nodes it points to are weak. To avoid such moving, we add self-edges for each node in the relational graph and we update A_{II} by (12), where $Ch_{\mathcal{G}}(I)$ is the set of I's child nodes in $\mathcal{G}$.

$$A_{II} = \max\left(0, \max_{M \in Ch_{\mathcal{G}}(I)} A_{MI} - \max_{N \in Pa_{\mathcal{G}}(I)} A_{IN}\right) \tag{12}$$

Then, we convert A to a probability transition matrix P by (13), which is used for Personalized PageRank in the following phase.

$$P_{IJ} = \frac{A_{IJ}}{\sum_O A_{IO}} \tag{13}$$

2) Node Ranking: A preference vector $\boldsymbol{u}$ is needed for Personalized PageRank to avoid a local trap [21]. We define $\boldsymbol{u}$ as (14), where X is a node in the relational graph, x is a metric within node X, and K is the key metric group.

$$u_X = \begin{cases} 0, & X = K \\ \max \textit{Correlation}(x, k), x \in X, k \in K, & X \neq K \end{cases} \tag{14}$$

As for the node representing the key metric group, $\boldsymbol{u}$ is set to zero, for the reason that the key metrics are almost impossible to be its own root cause. When it comes to the node representing the infrastructure metric group, we calculate $\boldsymbol{u}$ through the correlation between each infrastructure metric in this node and each key metric in the key metric group. In one failure, the trends of anomaly metrics are similar so that the correlation can be utilized as one of the foundations for root cause judgment. The data length used for $\textit{Correlation}(x, k)$ is $\ell_{corr} + \ell_{test}$ where ℓ_{corr} denotes the historical data length for correlation coefficient calculation. Considering that when a failure occurs, it will take a certain period of time for the failure to propagate from the root cause to the key metrics. Therefore, when calculating $\textit{Correlation}(x, k)$, we actively keep the time window of key metrics stationary and shift that of infrastructure metrics to the left step by step and then calculate the correlation. That is, for $0 \leq i \leq \ell_p$, we calculate the Pearson correlation coefficient of x with the time period $[t_{alert} - \ell_{corr} - i, t_{alert} + \ell_{test} - i]$ and k with the time period $[t_{alert} - \ell_{corr}, t_{alert} + \ell_{test}]$ in a loop, where ℓ_p is the maximum failure propagation time and t_{alert} is the start time of the alert as stated in Sec. III-A. The maximum value obtained is regarded as the final $\textit{Correlation}(x, k)$.

After we obtain the probability transition matrix P and the preference vector $\boldsymbol{u}$, the Personalized PageRank algorithm

can be expressed as (15), where $c \in (0, 1)$ is the damping parameter for PageRank, which is usually set to 0.85 [21]. A solution $\boldsymbol{v}$ to (15) serves as the node ranking score τ_X for each node X in the causal graph.

$$\boldsymbol{v} = cP^T\boldsymbol{v} + (1-c)\boldsymbol{u} \tag{15}$$

With τ_X, we can calculate the final score $\mathcal{R}$ with $\mathcal{R}_X = (1-\lambda)\alpha_X + \lambda\tau_X$ for each infrastructure node X (*i.e.*, metric group) in the causal graph, where λ is the weight coefficient. By sorting the final scores of these infrastructure metric groups in descending order, suspicious root cause metric groups are obtained and presented to the DBAs with all possible propagation paths extracted from the constructed causal graph. These paths can provide explanations for root cause localization, which can help DBAs conduct faster troubleshooting and determine failure mitigation actions more efficiently.

IV. Experiment

A. Evaluation Setup

1) Dataset: The experimental data is collected from an Oracle database in the production system of a large bank with a three-month period to evaluate *CauseRank* in practice. It contains 97 real-world failures, which fall into 9 types. The metrics are collected via private monitoring software configured by DBAs extracted from the statistics generated by Oracle, including 1 key metric and 233 infrastructure metrics. These metrics are grouped into 1 key metric group and 26 infrastructure metric groups based on DBAs' experience, respectively. The group-level root causes of these failures are labeled by DBAs according to the actual failure propagation situation. Existing domain knowledge used to construct the expert causal graph in the edge inference phase in this bank system is sorted out by experienced DBAs according to the relations and dependencies in real scenarios.

2) Evaluation Metrics: To quantify the performance of root cause localization, following the existing works [11], [14], [22], we adopt top-k accuracy ($A@k$) and mean average rank (MAR) for evaluation. $A@k$ represents the proportion of the failures whose top-k candidate suspicious root cause metric groups contain actual root cause metric groups. MAR is the mean of the average rank of the root cause metric groups in the suspicious metric groups of all the failures, which can more accurately describe the overall performance of the approach. The smaller the better. Specifically, MAR represents the average number of suspicious metric groups that DBAs need to examine in order to deal with one failure. When an actual root cause metric group is not in the suspicious metric groups, we apply mathematical expectations to estimate the number of metric groups that DBAs have to check additionally, that is, half the length of the remaining metric groups.

3) Baselines: We thoroughly compare our proposed *CauseRank* with several root cause localization approaches, including pattern matching-based, anomaly degree-based and causal analysis-based approaches. The chosen approaches have proven to outperform other existing work such as MicroScope [13] and DBSherlock [1], which will not be considered in our experiments. Notice that *CauseRank* without domain knowledge is also conducted for a fair comparison.

TABLE II
Comparison with Baselines

Method	$A@1$	$A@3$	$A@5$	MAR	$\uparrow MAR$	T. (s)
iSQUAD	0.372	0.698	0.744	3.21	33.6%	0.77
FluxRank	0.206	0.495	0.701	4.44	52.0%	**0.42**
CRD	0.183	0.394	0.542	6.11	65.1%	16.94
MicroCause	0.270	0.600	0.773	3.95	46.1%	958.80
*CauseRank**	0.412	0.756	0.835	2.90	26.6%	11.16
CauseRank	**0.557**	**0.825**	**0.938**	**2.13**	-	12.58

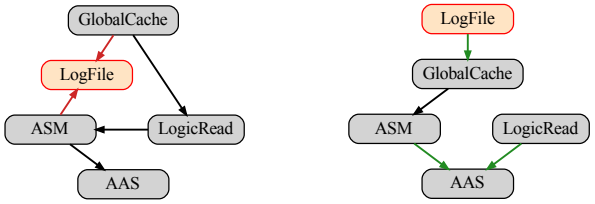
Domain knowledge is not used in the baseline methods (including *CauseRank**). The rightmost column shows the localization time per failure for each method.

- iSQUAD [3] allows DBAs to point out the root cause of each type of failure clustered by TOPIC, and then match new queries with each failure type online for failure diagnosis in cloud databases. We split each type of failure data into a training set and a testing set with the proportion of 60% and 40%, respectively. The training set is used for clustering and learning historical failure patterns. The experimental results are evaluated on the testing set.
- FluxRank [9] employs KDE (Kernel Density Estimation) to detect anomaly infrastructure metrics and localizes the root cause based on anomaly degrees.
- CRD [23] proposes to rank causal anomalies in a two-phase manner, in which a diffusion-based network reconstruction model is used to backtrack causal anomalies.
- MicroCause [11] proposes to detect anomaly metrics with SPOT [24], build a causal graph with these metrics by PCTS algorithm, and finally utilize TCORW for ranking.
- *CauseRank** is a variant of *CauseRank*, which infers the causal relations without any domain knowledge under the circumstance that other stages remain unchanged.

4) Experimental Details: We conduct all the experiments on a server with a 22-core 2.40GHz CPU (Intel(R) Xeon(R) CPU E5-2620 v3) and 57GB RAM. All the methods are implemented in Python, with which we can make a fair comparison of the speed. Based on the expert experience and the tolerance for failure localization of DBAs as well as the inherent characteristics of our dataset, we set $\ell_p = 5\text{mins}$, $\ell_{train} = 120\text{mins}$, $\ell_{test} = 10\text{mins}$, $\ell_{graph} = 7200\text{mins}$, $\ell_{corr} = 20\text{mins}$. As for the baseline methods, since the anomaly detection and causal inference are in a single metric level, we use the raw metric data (*i.e.* without group information) as input. The results of these methods are changed into group form according to the first occurrence of metrics belonging to each group, which are in the same form with the results of *CauseRank* for evaluation. For methods with randomness (*i.e.* CRD and MicroCause), we repeat the experiments 5 times to eliminate deviations.

B. Overall Performance

Table II shows the comparison results between *CauseRank* and the baseline methods. In addition to the above evaluation metrics, we also collect the average localization time per failure to evaluate the availability in the real-time scenario.



(a) Without domain knowledge (b) With domain knowledge

Fig. 4. Case study of domain knowledge. The red node denotes the root cause. Green edges denote edges inferred with domain knowledge and red edges denote edges conflict with domain knowledge.

The results show that *CauseRank* performs better than the baseline methods significantly with acceptable localization time regarding various evaluation metrics.

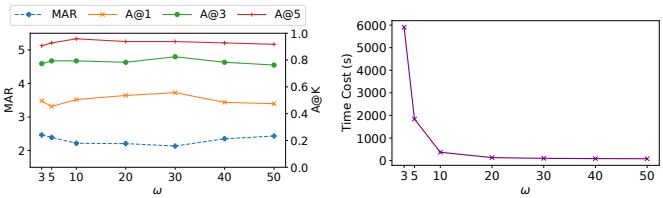
When implementing iSQUAD, we ensure that every failure in the test set can be found similar in the training set. As a result, its experimental results are better than other baselines. However, iSQUAD mainly focuses on the diagnosis of Intermittent Slow Queries. It is still difficult to be extended to the localization of the entire database system due to a large number of metrics and complex internal relations. Compared with FluxRank, our proposed approach takes the causal relations among the metric groups into consideration, which is crucial in accurate root cause localization. CRD requires the data to be of high quality since it is difficult to fit AutoRegressive eXogenous models in case of numerous missing data points. Furthermore, when localizing, CRD finds out the root cause by comparing the invariant network obtained from previous training with the broken network generated at the time point that the failure occurs. It does not consider the propagation time of failures, *i.e.*, the anomaly time of key metrics is not always the same as that of the root cause metrics. Thus, in our scenario, CRD has the worst performance. MicroCause tries to infer the causal relations to get the failure propagation path. However, the relations among metrics often receive the interference of many irrelevant factors, leading to a complex and unreliable causal graph. Besides, the causal discovery algorithm PCTS used in MicroCause suffers from long running time, resulting in unavailability in online systems.

The results also show that *CauseRank** outperforms other baseline methods, even without domain knowledge, which proves the effectiveness of the proposed G-GES and COPP in the root cause localization task. However, due to the high similarity of the metrics, critical failure propagation paths may be constructed incorrectly, which greatly affects the accuracy of the results. In contrast, the proposed *CauseRank* can effectively take advantage of domain knowledge and achieve the best performance in our root cause localization task.

We conduct a case study for detailed explanations. Fig. 4 shows a part of the causal graphs inferred by G-GES without and with domain knowledge, respectively. Actually, the fluctuations of the metrics in "GlobalCache", "LogFile", and "ASM" are similar in this failure, which makes it difficult for G-GES to infer the correct causal relations among them. In Fig. 4a, the causal relations among these three metric groups conflict with those provided by DBAs. As a result, *CauseRank* with such a graph ranks "GlobalCache" at the first by mistake. In contrast, the causal graph inferred with domain knowledge (Fig. 4b) shows correct causal relations among them, resulting in higher ranking accuracy. This case shows that our integration strategy of domain knowledge and G-GES is practical and effective.

TABLE III
ABLATION STUDY OF KEY TECHNIQUES IN *CauseRank*

Method	$A@1$	$A@3$	$A@5$	MAR	$\uparrow MAR$
CauseRank w/o Grouping	0.381	0.742	0.867	2.897	26.3%
CauseRank w/o G-GES	0.412	0.732	0.900	2.660	19.8%
CauseRank w/o COPP	0.495	0.763	0.928	2.289	6.8%
CauseRank	**0.557**	**0.825**	**0.938**	**2.134**	-



(a) Performance under different ω (b) Time cost under different ω

Fig. 5. Study of ω

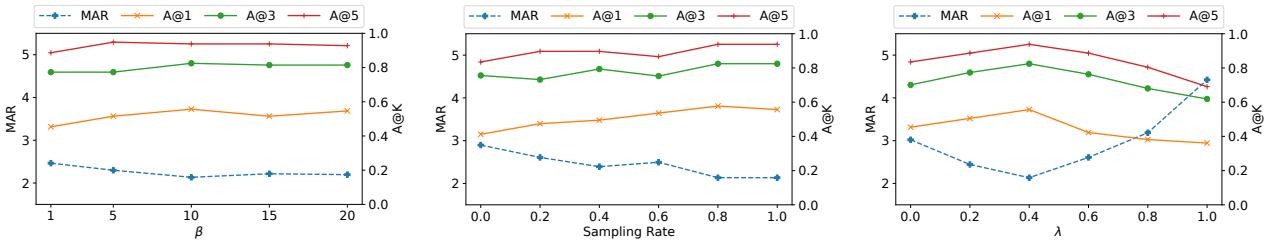
C. Ablation Study

To investigate the contributions of key techniques of *CauseRank*, we perform several additional experiments for ablation study, in which several variants of *CauseRank* are constructed. We have proved the important role of domain knowledge in Sec. IV-B and it will not be explained in this part. Table III summarizes the overall ablation study results. As shown in Table III, all of the key techniques in *CauseRank* have positive effects on the results of root cause localization, which proves these techniques are of great necessity.

1) Impact of Grouping: We remove the metric grouping technique of *CauseRank*, which means that naive GES is applied in the causal graph building stage, and (11) is calculated with scoring criterion S. This variant performs poorly compared with the original *CauseRank* method as shown in Table III, which further demonstrates the advantages of *CauseRank* in dealing with numerous metrics by metric grouping.

2) Impact of G-GES: Intuitively, the typical failure propagation paths in the expert causal graph inspired by domain knowledge can help us get accurate causal graphs. To examine the actual performance of G-GES, we only use the expert causal graph for root cause ranking. Results in Table III show a huge improvement in using the causal graph constructed by domain knowledge enhanced G-GES, indicating the importance of the temporary causal discovery algorithm in root cause localization, even with DBAs' domain knowledge.

3) Impact of COPP: To verify our proposed ranking algorithm COPP, we modify (11) by using the maximum Pearson correlation coefficient between the metrics of adjacent nodes as weights in the graph setup phase. From Table III, we can conclude that COPP has a certain improvement on the overall performance, especially on A@1.



(a) Performance under different β (b) Performance under different sampling rate (c) Performance under different λ

Fig. 6. Study of parameters

D. Study of Parameters

1) Evaluation of ω: In G-GES, we use the parameter ω as the penalty coefficient to limit the number of parents of each node as shown in (8). By applying this penalty, we can obtain more significant failure propagation paths and reduce the interference of noise on the construction quality of causal graphs. To investigate the influence of ω, we conduct several experiments with different values of ω. As shown in Fig. 5, the performance of *CauseRank* does not fluctuate much with the change of ω. The results indicate that even if the number of edges is limited, the pivotal failure propagation paths can also be inferred by G-GES enhanced with domain knowledge so as to achieve good performance in root cause localization. Moreover, regarding the GFS step of G-GES in causal graph building, the limit of graph density greatly improves the speed. Fig. 5b shows the time costs for running *CauseRank* with different ω. It can be found that the time cost decreases rapidly as ω increases. In *CauseRank*, we set ω to 30 in order to achieve high performance with reasonable time consumption.

2) Impact of β: β determines the impact degree of domain knowledge in *CauseRank*, which should be set to an appropriate value. Experimental results of *CauseRank* with different β are depicted in Fig. 6a. From the results, we can conclude that when β is set too small, domain knowledge will only have little effect on the inference of causal graphs, resulting in slightly poor performance. When $\beta >= 10$, *CauseRank* is not sensitive to the parameters. In order to ensure the generalization of the algorithm, we set β to 10 in the experiments.

3) Impact of #Rules in Domain Knowledge: In practical applications, the number of expert rules in domain knowledge is limited by DBAs' empirical knowledge and time spent on constructing rules. Therefore, in this part, we investigate the effect of #rules on overall performance. We sample the expert rules at different proportions randomly and obtain the results of root cause localization separately with the sampled rules as input. Results of different sampling proportions are shown in Fig. 6b. It can be found that the accuracy of root cause localization shows an increasing trend with the increase of the number of expert rules, indicating the advantages of integrating domain knowledge into the construction of causal graphs.

4) Impact of λ: λ controls the weight of τ_X in calculating the final root cause ranking score. Results demonstrated in Fig. 6c indicate that the performance reaches the best when $\lambda = 0.4$. Compared with the results of only adopting the anomaly degrees (*i.e.* $\lambda = 0.0$), the overall performance of *CauseRank* is significantly improved by taking causal graphs into consideration, which shows the crucial contribution of the causal information.

V. Related Work

In this section, we discuss the related work on root cause analysis algorithms and causal discovery algorithms.

Root Cause Analysis Several works [5], [6] localize the root cause through supervised models. MEPFL [5] converts the localization problem into a multi-classification problem and employs a machine learning model to solve it. Some works take advantage of failure pattern matching to localize failures [3], [7], [8]. Fingerprint [7] characterizes the system status through analyzing metrics and identifies the performance crises by comparing the current status with the historical failure status. iSQUAD [3] performs online diagnosis by matching the pattern of each failure type consistent with that of a new query, where failure types are clustered and labeled by DBAs offline. However, these methods generally rely on historical failure data, even annotated, which are not always available in reality and cannot handle new-type failures.

In order to improve the versatility and flexibility of the approaches, many researchers focus on anomaly detection and causal analysis to address the localization problem. FluxRank [9] ranks the potential root cause according to the anomaly degree of the metrics. ϵ-Diagnosis [10] also adopts an anomaly detection-based method to localize a specific type of microservice failure. While the determination of root cause metrics cannot rely solely on the anomaly degree. Dbsherlock [1] requires the end-user to select the region of unexpected performance to provide support for its predicates analysis and mark the root cause metric to construct causal models for the root cause ranking in online scenarios, which requires too many end-user instructions. MonitorRank [14] adopts the random walk algorithm on the calling graph generated by Hadoop for root cause ranking, which is not that suitable for our scenario because the causal relations of metrics are unknown in database systems. MicroScope [13] finds the metrics related to the failure through traversing the causal graph built by the parallelized PC algorithm and uses the metric similarity as the ranking score. CauseInfer [15] builds the metric causal graph through PC algorithm and uses DFS to obtain the possible root cause metrics. However, when the causal graph is not ensured to be accurate, the metric represented by the root

node is not always the root cause. ExplainIt [25] performs root cause analysis by sorting the candidate causal hypotheses identified by the operators, which requires heavy reliance on operators. MicroCause [11] proposes PCTS for constructing causal graphs more accurately, but it is so time-consuming that it cannot be adapted to the online scene.

Causal Discovery Existing works usually apply constraint-based algorithms to build the causal graphs, such as PC [15], parallelized PC [13], and PCTS [11] algorithms. These constraint-based algorithms [26], [27] mostly rely on conditional independence tests. However, conditional independence tests are not guaranteed to be correct due to the noise [28], [29], especially when it is applied on a large-scale time-series dataset, leading to high dimensionality and low detection power [27]. The score-based algorithms [19], [30] build the causal graph by optimizing the global score of the graph, which can overcome the shortcomings mentioned above.

VI. Conclusion

Performance diagnosis in OLTP database systems is a critical task for maintaining service availability and user experience. The primary requirements and challenges are accuracy and efficiency, especially in the face of a large number of metrics. In this paper, we propose *CauseRank*, which utilizes metric group analysis to provide high-quality failure localization. Core techniques in *CauseRank* are a causal discovery algorithm (G-GES) integrating domain knowledge and a ranking algorithm (COPP). Extensive experiments on 97 real-world failures demonstrate that *CauseRank* outperforms the baseline methods in terms of accuracy and the time cost of localization can meet the requirements of online diagnosis. Besides, the framework of *CauseRank* is generic to other large-scale system components, *e.g.*, cloud databases, middleware, load balancers, and web servers. We believe that *CauseRank* can also achieve good performance in other root cause localization tasks.

Acknowledgment

This work is supported by the National Key R&D Program of China 2019YFB1802504, the State Key Program of National Natural Science Foundation of China (Grant No. 61902200 and 62072264), the China Postdoctoral Science Foundation (2019M651015).

References

[1] D. Y. Yoon, N. Niu, and B. Mozafari, "DBSherlock: A performance diagnostic tool for transactional databases," in *SIGMOD*, 2016.

[2] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood, "Automatic performance diagnosis and tuning in oracle," in *CIDR*, 2005.

[3] M. Ma, Z. Yin, S. Zhang, S. Wang, C. Zheng, X. Jiang, H. Hu, C. Luo, Y. Li, N. Qiu, F. Li, C. Chen, and D. Pei, "Diagnosing root causes of intermittent slow queries in cloud databases," in *VLDB*, 2020.

[4] R. Cornejo, "Statistical analysis," in *Dynamic Oracle Performance Analytics*. Apress, 2018, pp. 61–77.

[5] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *ESEC/FSE*, 2019.

[6] P. Dogga, K. Narasimhan, A. Sivaraman, S. K. Saini, G. Varghese, R. N. Ucla, P. University., Nyu, A. Research, and India, "Revelio: Ml-generated debugging queries for distributed systems," *ArXiv*, vol. abs/2106.14347, 2021.

[7] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: Automated classification of performance crises," in *EuroSys*, 2010.

[8] Á. Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez-Hernández, and V. Muntés-Mulero, "Graph-based root cause analysis for service-oriented and microservice architectures," *J. Syst. Softw.*, vol. 159, 2020.

[9] P. Liu, Y. Chen, X. Nie, J. Zhu, S. Zhang, K. Sui, M. Zhang, and D. Pei, "FluxRank: A widely-deployable framework to automatically localizing root cause machines for software service failure mitigation," in *ISSRE*, 2019.

[10] H. Shan, Y. Chen, H. Liu, Y. Zhang, X. Xiao, X. He, M. Li, and W. Ding, "ϵ-diagnosis: Unsupervised and real-time diagnosis of small- window long-tail latency in large-scale microservice platforms," in *WWW*, 2019.

[11] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia, Z. Wang, and D. Pei, "Localizing failure root causes in a microservice through causality inference," in *IWQoS*, 2020.

[12] L. Wu, J. Tordsson, J. Bogatinovski, E. Elmroth, and O. Kao, "Microdiag: Fine-grained performance diagnosis for microservice systems," in *Proceedings of the IEEE/ACM International Workshop on Cloud Intelligence*, 2021.

[13] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *ICSOC*, 2018.

[14] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," in *SIGMETRICS*, 2013.

[15] P. Chen, Y. Qi, P. Zheng, and D. Hou, "Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems," in *INFOCOM*, 2014.

[16] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker, "Staring into the abyss: An evaluation of concurrency control with one thousand cores," *VLDB*, vol. 8, no. 3, p. 209–220, 2014.

[17] K. R. Dittrich, S. Gatziu, and A. Geppert, "The active database management system manifesto: A rulebase of adbms features," in *Proceedings of the Rules in Database Systems*, 1995.

[18] Y. Sheng, A. Tomasic, T. Zhang, and A. Pavlo, "Scheduling oltp transactions via learned abort prediction," in *Proceedings of the Workshop on aiDM*, 2019.

[19] D. M. Chickering, "Optimal structure identification with greedy search," *J. Mach. Learn. Res.*, vol. 3, pp. 507–554, 2002.

[20] M. Ma, W. Lin, D. Pan, and P. Wang, "MS-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications," in *ICWS*, 2019.

[21] G. Jeh and J. Widom, "Scaling personalized web search," in *WWW*, 2003.

[22] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *CCGrid*, 2018.

[23] J. Ni, W. Cheng, K. Zhang, D. Song, T. Yan, H. Chen, and X. Zhang, "Ranking causal anomalies by modeling local propagations on networked systems," in *ICDM*, 2017.

[24] A. Siffer, P. Fouque, A. Termier, and C. Largouët, "Anomaly detection in streams with extreme value theory," in *SIGKDD*, 2017.

[25] V. Jeyakumar, O. Madani, A. Parandeh, A. Kulshreshtha, W. Zeng, and N. Yadav, "Explainit! – a declarative root-cause analysis engine for time series data," in *SIGMOD*, 2019.

[26] M. Kalisch and P. Bühlmann, "Estimating high-dimensional directed acyclic graphs with the PC-algorithm," *J. Mach. Learn. Res.*, vol. 8, pp. 613–636, 2007.

[27] J. Runge, P. Nowack, M. Kretschmer, S. Flaxman, and D. Sejdinovic, "Detecting and quantifying causal associations in large nonlinear time series datasets," *Science Advances*, vol. 5, no. 11, p. eaau4996, 2019.

[28] R. D. Shah and J. Peters, "The hardness of conditional independence testing and the generalised covariance measure," *The Annals of Statistics*, vol. 48, no. 3, pp. 1514 – 1538, 2020.

[29] K. Zhang, J. Peters, D. Janzing, and B. Schölkopf, "Kernel-based conditional independence test and application in causal discovery," in *UAI*, 2011.

[30] J. Ramsey, M. Glymour, R. Sanchez-Romero, and C. Glymour, "A million variables and more: the fast greedy equivalence search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images," *International Journal of Data Science and Analytics*, vol. 3, pp. 121–129, 2016.