FluxInfer: Automatic Diagnosis of Performance Anomaly for Online Database System

Ping Liu [§‡], Shenglin Zhang [†], Yongqian Sun* [†], Yuan Meng [§‡], Jiahai Yang [‖‡¶], Dan Pei [§‡]
[§]Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
[‖]Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China
[†]College of Software, Nankai University, Tianjin 300071, China
[¶]Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen 518000, China
[‡]Beijing National Research Center for Information Science and Technology (BNRist)

***Abstract*—The root cause diagnosis of performance anomaly for online database anomalies is challenging due to diverse types of database engines, different operational modes, and variable anomaly patterns. To relieve database operators from manual anomaly diagnosis and alarm storm, we propose FluxInfer, a framework to accurately and rapidly localize root cause related KPIs for database performance anomaly. It first constructs a Weighted Undirected Dependency Graph (WUDG) to represent the dependency relationships of anomalous KPIs accurately, and then applies a weighted PageRank algorithm to localize root cause related KPIs. The testbed evaluation experiments show that the AC@3, AC@5, and Avg@5 of FluxInfer are 0.90, 0.95, and 0.77, outperforming nine baselines by 64%, 60%, and 53% on average, respectively.**

***Index Terms*—performance anomaly, root cause localization, KPI**

I. Introduction

The performance of today's database system is vitally important to Internet services, since a performance issue of database can degrade service performance and in turn impact user experience. Therefore, the performance monitoring and anomaly root cause analysis of database systems are crucial to these services. However, they are challenging due to diverse types of database engines, different operational modes, and variable anomaly patterns.

To rapidly diagnose anomalies and trigger mitigation, database operators monitor hundreds of KPIs (Key Performance Indicator) of each database instance. Usually, when a database anomaly (*e.g.*, the database response times are too slowly) arises, some of these KPIs manifest anomalous patterns (say, sudden changes). Operators thus can proactively detect database anomalies and conduct root cause analysis according to these KPIs. We take a real-world example to elaborate how operators *manually* analyze anomaly root causes. Figure 1 shows some anomalous KPIs when a database becomes unavailable. 78 KPIs becomes anomalous in this database anomaly. Some anomalous KPIs indicate the root cause of this database anomaly and thus is closely related to the root cause (called **root cause related KPI** hereafter), while other anomalous KPIs are not directly related to the root cause

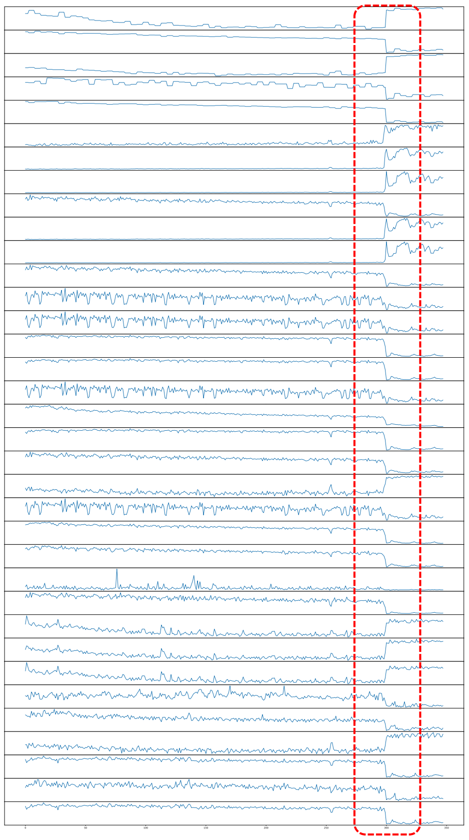

Fig. 1: Some anomalous KPIs during a real-world database anomaly.

(called **symptom KPI** hereafter). In this case, for example, a sudden increase in workload leads to this database anomaly. Therefore, QPS (a KPI), which represents the workload of the database, is a root cause related KPI. On the other hand, the increase in workload also leads to a sudden increase in CPU utilization, which is represented by the KPI of CPU_USAGE. Consequently, CPU_USAGE is a symptom KPI.

Manual anomaly diagnosis. When an anomaly occurs, operators usually take two steps to manually localize the root cause related KPIs from such a large number of anomalous KPIs: (1) operators manually infer the dependency relationships of these KPIs, which is highly dependent on the domain knowledge and experience; (2) they then localize the

*Yongqian Sun is the correspondence author.

root cause related KPIs based on the dependency relationships. However, it is time-consuming to manually diagnose a database anomaly even for a highly experienced operator, causing the services to suffer from performance degradation for a long period. In addition, hundreds of database anomalies happen in a large online database cluster [1], and manually diagnosing such a large number of anomalies is tedious, error-prone and unscalable.

Alarm storm. Typically, an alarm is generated and sent to operators when a KPI is detected anomalous. Since a single database anomaly can trigger up to hundreds of these alarms, a large online database cluster housing tens of thousands to millions of database systems can produce tens of thousands of such alarms per day, which is called alarm storm. Alarm storm has always been a challenge for operators because such a large number of alarms can degrade diagnosis efficiency. Although operators can avoid alarm storm by adjusting alarm filtering rules (*e.g.*, tuning the thresholds of anomaly detection algorithms) for different database instances, it is still a great deal of work to find the best rule for each database instance.

To relieve operators from manual anomaly diagnosis and alarm storm, we try to automate anomaly diagnosis of performance anomaly for online database systems. In this way, we can automatically localize the root cause related KPIs soon after a database anomaly arises to quickly mitigate the loss. Moreover, operators can focus on these KPIs rather than symptom KPIs, successfully addressing the burden of alarm storm. Although DBSherlock [2] was proposed as a tool to assist operators in explaining performance anomalies of database systems, it did not *directly* localize the root cause or root cause related KPIs. Operators have to manually combine DBSherlock's output with domain knowledge to localize the root cause KPIs. Accordingly, we propose FluxInfer to automatically, accurately, and rapidly localize root cause KPIs to diagnose performance anomaly for online database systems. However, FluxInfer faces the following two challenges.

Challenge 1: Accurately represent the dependency relationships of KPIs. Motivated by the manual diagnosis process of operators, we find that inferring the dependency relationships of KPIs is crucial to automatically diagnose anomalies. Some state-of-the-art works [3]–[7] tried to automatically construct a directed acyclic graph (DAG) using PC algorithm [8] to represent the dependency relationships of KPIs. However, due to the unknown latent variables which are hardly observed and inferred, the constructed DAG can generate incorrect dependency relationships, which tends to cause incorrect localization results (details are discussed in §II). To address this challenge, we propose an algorithm to automatically construct a Weighted Undirected Dependency Graph (WUDG) to accurately represent the dependency relationships of anomalous KPIs. To the best of our knowledge, this is the first time that WUDG is used for anomaly diagnosis of computer systems. *We believe that WUDG can be used to accurately infer the dependency relationships of KPIs for more anomaly diagnosis scenarios beyond online database systems.* **This is the first contribution of this paper.**

Challenge 2: Automatically localize the root cause related KPIs based on a WUDG. Some state-of-the-art

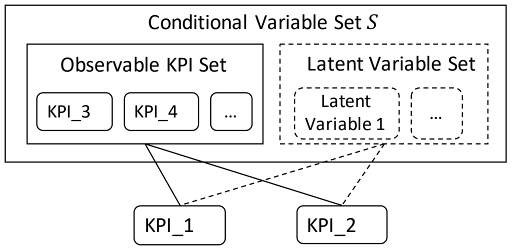

Fig. 2: Explanation of $I(\text{KPI\_1}, \text{KPI\_2}|S)$.

works [3]–[7] proposed to use Deep-First Search, Random Walk, *etc.*, to traverse DAG for localizing root cause. However, these localization algorithms can only be used in the *directed* dependency graphs, and they are inapplicable to the *undirected* WUDG. To address this challenge, we propose to use a weighted PageRank algorithm to traverse WUDG [9], which can accurately localize root cause related KPIs. **This is the second contribution of this work.**

Detailed evaluation experiments on our testbed show that the AC@3, AC@5, and Avg@5 of FluxInfer are 0.90, 0.95, and 0.77, outperforming nine baselines by 64%, 60%, and 53% on average, respectively. Additionally, the average diagnosis time of FluxInfer is 53 seconds, which is approriate for online diagnosis. **This is the third contribution of this work.**

II. Representation challenge

Inferring the dependency relationships among KPIs is crucial to automatically diagnose anomalies. Most of the state-of-the-art works [3]–[7] tried to automatically construct a Directed Acyclic Graph (DAG) by PC algorithm [8] to represent the dependency relationships among KPIs. Then the root causes are localized by traversing the DAG with different algorithms. However, PC algorithm may infer some incorrect dependency relationships due to the unknown latent variables, which will lead to incorrect localization results. In this section, we first introduce the details of the PC algorithm. Then we explain the representation challenge of the constructed DAG.

A. A common method: PC algorithm

PC algorithm [8] is the most popular algorithm to construct a causal graph from observational data. It assumes *faithfulness*, which means that there is a directed acyclic graph, G, such that the causal relationships among random variables are exactly those represented by G. Its input is the observed values of N random variables. PC algorithm will output a DAG with N nodes, where each node represents one random variable. There are four steps in the PC algorithm:

1) Construct a fully connected undirected graph of the N random variables (all nodes are connected).
2) Perform a conditional independence test on each adjacent variables (X and Y) conditional on a variable set S, denoted by $I(X, Y|S)$. If a conditional independence exists, the edge between the two variables is removed. In this step, the size of the conditional variable set S increases step by step until there are no more variables that can be added into S.
3) Determine the directions of some edges on the basis of V-structure [10]. A V-structure is a condition to decide

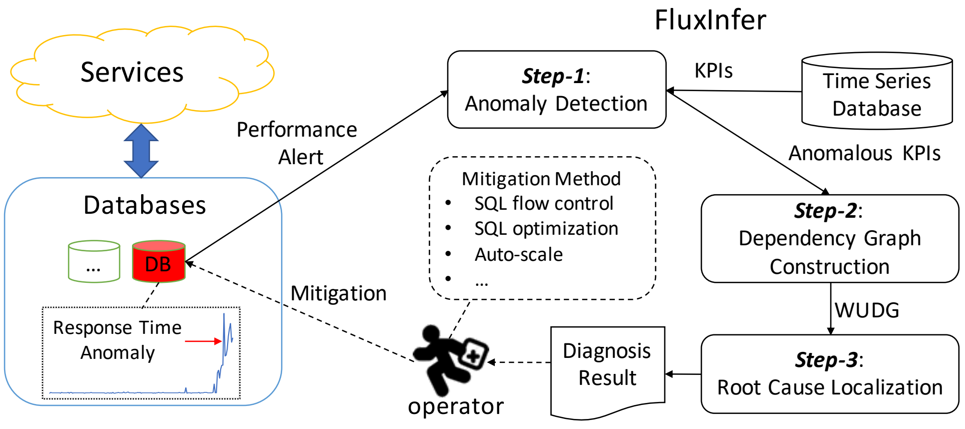

Fig. 3: The overview of FluxInfer

the direction of an edge. Suppose three nodes X, Y, Z be a part of a graph. X and Y are connected, and Y and Z are connected (X-Y-Z). One obtains a causal relationship X $\rightarrow$ Y $\leftarrow$ Z if X and Z are not conditionally independent for Y.

4) Determine the direction of the rest of the edges with orientation rules.

B. Representation Challenge

As described in the details of the PC algorithm, if the set of independencies (step-2 and step-3 of PC algorithm) is faithful to a graph and we have a perfect way of determining whether $I(X, Y|S)$, then the PC algorithm has a guarantee of producing a graph equivalent to the original one. However, none of these conditions is verified. Firstly, the conditional independence test is a statistical test that may have errors. Secondly, the conditional variable set S cannot be completely observed. In practice, the conditional variable set S may include some latent variables that are hard to be observed or inferred, which will produce some incorrect relationships in the DAG. As shown in Figure 2, the latent variable set could affect the conditional independence test of $I(\text{KPI\_1}, \text{KPI\_2}|S)$. During evaluation, we found that PC-algorithm inferred some incorrect dependency relationship. For example, NET_SEND$\rightarrow$ QPS. QPS indicates the workload and NET_SEND indicates the number of bytes sent by the database. Obviously, QPS is not dependent on NET_SEND.

III. System Overview and Current Limitation

Figure 3 shows an overview of FluxInfer. The services need databases to support their mission-critical and real-time applications. FluxInfer is triggered by the performance anomaly alert, for example, the response time of a database suddenly increases. FluxInfer contains three steps: *Anomaly Detection*, *Dependency Graph Construction* and *Root Cause Localization*. Here we will give a cursory introduction of them.

Step-1: Anomaly Detection. All KPIs of an anomalous database are detected in this step. The KPI values around the alert time are obtained from the time series database which stores historical KPI values of databases. Due to the real world's KPI values are noisy and often fluctuate regardless of system failure, a jumble of noises, normal fluctuations and anomalous changes will affect the performance of anomaly detection. Therefore, we designed a cluster-based anomaly detection algorithm that can detect anomalies robustly.

Step-2: Dependency Graph Construction. The input of this step is the anomalous KPIs detected by Step-1. A weighted undirected dependency graph is constructed in this step, which can accurately represent the dependency relationships among the anomalous KPIs.

Step-3: Root Cause Localization. The purpose of this step is to localize the root cause related KPIs based on the dependency graph constructed in step-2. We designed a localization algorithm based on the weighted PageRank [9] algorithm to analyze the dependency graph, and finally recommends the ranking list of possible root cause related KPIs.

After localizing the root cause related KPIs, operators can quickly take actions to mitigate the database anomalies based on the diagnosis results of FluxInfer, like SQL flow control, SQL optimization, auto-scale, *etc.*. Next, we will introduce the limitation of FluxInfer.

FluxInfer diagnoses the anomalies whose root cause can affect at least one of the KPIs. It is straightforward to see the reason: if the root cause does not manifest itself in any of the KPIs, FluxInfer has no means of distinguishing between root cause related KPIs and symptom KPIs. For example, mostly SELECT statements are executed slowly if the proper indexes are not created on tables. In this scenario, although some KPIs will also show anomalous patterns, the root cause (poor table design) can not *directly* manifest itself in any of the KPIs.

IV. System Design

A. Robust Anomaly Detection

The real world's KPI values are noisy and often fluctuate regardless of system anomaly. So the KPI values needs to be smoothed before anomaly detection. However, the commonly used data smoothing algorithms (*e.g.*, Kalman Filter, Moving Average, *etc.*) cannot be applied in our scenario. When a system is anomalous, the anomalous changes could be wrongly smoothed without distinction among a jumble of noises, normal fluctuations, and anomalous changes by these algorithms. Therefore, we designed a cluster-based robust anomaly detection algorithm. The core idea is that the clustering algorithm can divide KPI values into different segments, then the noisy segments, anomalous segments, and normal segments can be

Algorithm 1 Smoothing

```
Input:
    The values of a KPI, $\{x_i\}$;
Output:
    Smoothed data, $\{\grave{x}_i\}$
    Segments, $\{s_j\}$;
1: $\{\grave{x}_i\} \Leftarrow \{x_i\}$
2: while True do
3:     Cluster $\{\grave{x}_i\}$ into two clusters by Gaussian Mixture Model;
4:     $\{s_j\} \leftarrow$ Split $\{\grave{x}_i\}$ into $k$ segments by the clustering results;
5:     if $k <= 2$ then
6:         return $\{\grave{x}_i\}$, $\{s_j\}$;
7:     end if
8:     for $j = 1$; $j < k-1$; $j++$ do
9:         if $|s_j| < |s_{j+1}|$ then
10:            Replace $s_j$ with samples which are randomly sampled from $s_{j+1}$;
11:        end if
12:    end for
13:    if $\{\grave{x}_i\} \neq \{x_i\}$ then
14:        $\{x_i\} \leftarrow \{\grave{x}_i\}$
15:    else
16:        return $\{\grave{x}_i\}$, $\{s_j\}$;
17:    end if
18: end while
```

precisely divided. Next, we first introduce how to smooth the KPIs based on the clustering algorithm. Then we show how to detect anomalous KPIs based on the smoothing results.

1) Data Smoothing: Algorithm 1 shows the details of smoothing. It is a looping algorithm. During each loop, some noises are smoothed. The algorithm will loop multiple times until no data need to be smoothed. The input is the values of a KPI, denoted by $\{x_i\}$, and the smoothed values are denoted by $\{\grave{x}_i\}$. In each loop, firstly the KPI values are clustered into two clusters by Gaussian Mixture Model [11]. The reason for setting two clusters is: *The KPI values can be considered as normal and abnormal*. The abnormal cluster may include noisy values and anomalous values. Then the clustering results are used to split the KPI values into k segments (from 0 to $k-1$). Figure 4 shows an example of the segments. We use $\{s_j\}$ to denote the segment set, where s_j denotes the segment j. We also use $|s_j|$ to denote the length of s_j, which represents the number of data points in s_j. Due to the length of a noisy segment is smaller than its adjacent segments, s_j needs to be smoothed if the length of s_j is smaller than s_{j+1}: $|s_j| < |s_{j+1}|$. A segment s_j is smoothed by replacing its values with samples that are randomly sampled from s_{j+1}, as shown in Figure 4. The first segment s_0 is ignored as an invalid segment because it may only be part of a segment. Due to the last segment s_{k-1} is in the anomaly period, s_{k-1} does not need to be smoothed.

2) Anomaly Detection : After smoothing, some segments are obtained. Due to the last segment is in the database anomaly period, the last change may be related to the anomaly.

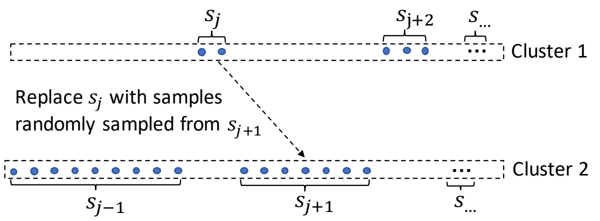

Fig. 4: An example of smoothing segment. The values of a KPI are clustered into two clusters. Then the clustering results are used to split the KPI values into different segments.

Therefore, we focus on the last change: between s_{k-2} and s_{k-1}. In our scenario, all KPIs (include normal KPIs and anomalous KPIs) are smoothed by the smoothing algorithm (§IV-A1). Therefore, the last change may be normal. So we need to find the anomalous changes by measuring the abnormality of the last change.

We use z-score to measure the abnormality of a change. The z-score of each data x_i in segment s_{k-1} is calculated by the $mean$ and standard deviation (std) of data in segment s_{k-2}: $\frac{x_i - mean}{std}$. Then the mean of z-scores of data in segment s_{k-1} can be used to represent the z-score of segment s_{k-1}, denoted by $\bar{z}$. Finally, the 3-sigma rule is used to detect anomalous changes. if $|\bar{z}| > 3 * std$, the change is anomalous. After anomaly detection, all anomalous KPIs will be used to construct the dependency graph (WUDG).

B. Dependency Graph Construction

To localize the root cause related KPIs, the dependency relationships among KPIs need to be constructed. Due to that the PC algorithm may infer some incorrect dependency relationships (§II), we proposed a Weighted Undirected Dependency graph (WUDG) which can accurately represent the dependency relationships among KPIs. The core idea of WUDG is: *if two KPIs have a dependency relationship, then the two KPIs are not independent*. Therefore, the design of WUDG based on whether the dependency relationships exist among KPIs (undirected dependency graph), instead of inferring the dependency directions among KPIs (directed dependency graph). It is more accurate to determine the dependency's existence than the direction of the dependency. Next, we will introduce the details of WUDG construction.

Firstly, we construct a complete graph for all anomalous KPIs. Secondly, the independence of all pairs on the complete graph is tested. Two algorithms are commonly used for the independence test [12]: G-square Test [13] and Fisher-Z Test [13]. G-square Test works with discrete values, and Fisher-Z works with continuous values. Due to all KPIs in our scenario are continuous values, we use Fisher-Z Test to test the independence among anomalous KPIs.

The Fisher-Z test evaluates independence on the basis of Pearson's correlation coefficient. This test is a combination of two statistical techniques: Fisher-Z transformation to estimate a population correlation coefficient, and a partial correlation to evaluate the effect of other nodes. We use X and Y to denote two KPIs, the statistic Z_s between X and Y is defined as:

$$Z_s = \frac{\sqrt{m-3}}{2} log \frac{1+r}{1-r},$$

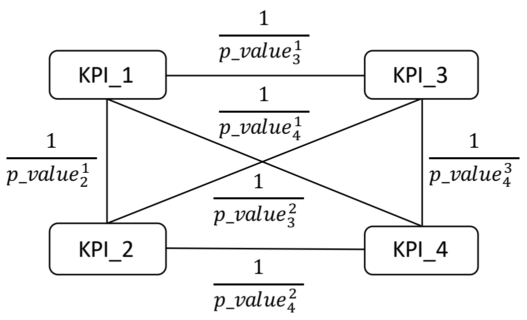

Fig. 5: An example of a WUDG. p_value$_m^n$ denotes the p_value of Fisher-Z Test between KPI n and m.

where m is the size of KPIs, and r is the partial correlation of X and Y.

The strength of the dependency between two KPIs can be measured by the p_value of the null hypothesis of the Fisher-Z Test. The smaller the p_value, the stronger the dependency relationship, and vice versa. Therefore, we set $\frac{1}{\text{p\_value}}$ to the weight of the corresponding edge. Finally, we construct a weighted undirected dependency graph (WUDG) to represent the dependency relationships among the anomalous KPIs. Figure 5 shows an example of a WUDG among four KPIs.

C. Root Cause Localization

The anomalous KPIs in the WUDG contain root cause related KPIs and symptom KPIs. Due to that the root cause of a database system can quickly spread and lead to more and more anomalous KPIs, the root cause has the largest influence on the anomaly spread network (the dependency graph). Therefore, we suppose that the root cause related KPI is the KPI who has the largest influence on the WUDG. The weighted PageRank algorithm have been used to measure the influences of nodes in a weighted undirected graph [9], [14]. For a KPI node u, we designed a weighted PageRank algorithm to compute a score $PR(u)$ for u:

$$PR(u) = (1-d) + d * \sum_{v \in B(u)} PR(v) * W_{(u,v)}^2,$$

where $B(u)$ denotes a set of KPI nodes that directly connect to u, $W_{(u,v)}$ denotes the weight of edge (u, v), and d is set to 0.85 which is a commonly used value [9]. Finally, all anomalous KPIs are ranked by their scores, and the possible root cause related KPIs are the KPIs ranked at the top.

V. Evaluation

In this section, we will introduce the details of the evaluation. The experiment setup, evaluation cases, and evaluation metrics are presented in §V-A, §V-B, and §V-C, respectively. FluxInfer is evaluated against nine state-of-the-art baseline approaches in §V-D. The design of robust anomaly detection is evaluated in §V-E. The diagnosis time of FluxInfer is discussed in §V-F. Finally, an interesting case is shown in §V-G.

A. Experiment Setup

We constructed a testbed to generate accurately labeled anomalies of database performance for evaluation. Figure 6 shows an overview of our testbed. We installed three Docker containers on a server that has 130G memory and 256 cores.

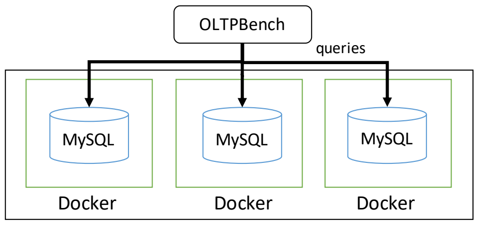

Fig. 6: The overview of Testbed.

MySQL database instance was deployed on each Docker container. We used the OLTPBenchmark framework [15] to simulate clients. The OLTPBenchmark framework was deployed on another server. We used stress-ng [16] to inject anomalies of CPU, IO, and memory. Each experiment consisted of 30 minutes of the normal state and different durations of abnormal state. We ran our experiments using TPC-C [17]. The default setting used in our TPC-C workload was a scale factor of 500 with 150 terminals. We also experimented with different scale factors (from 50 to 500) and the number of terminals (from 10 to 150). The results were consistent across these different settings. Table I shows the KPIs used for evaluation. These KPIs are standard KPIs from Linux OS, Docker, and MySQL. All the KPI data are collected at one-second intervals.

B. Evaluation cases

To evaluate FluxInfer, we injected five different types of anomalies to represent some of the important types of real-world problems that may deteriorate the performance of online databases. After the 30 minutes run of the normal workload in each case, we invoked the actual cause of an anomaly with different durations. Due to that our testbed needs to be rebooted after each anomaly injection for restoring the system to normal, we only collected 30 different cases for each type of anomaly by varying the duration. The duration of the anomalies ranged from 1 to 5 minutes with an increment of 10 seconds, yielding 30 cases for each type of anomaly (a total of 150 cases). Table II lists the types and descriptions of the different types of anomalies in our evaluation. These anomalies are designed to reflect a wide range of realistic scenarios that can negatively impact the performance of online databases.

C. Evaluation Metric

To evaluate the performance of each algorithm on a set of anomaly cases A, we use two metrics: $AC@k$ and $Avg@k$. These two metrics are the most commonly used metrics in the evaluation of the ranking results of the root cause localization algorithms in recent works [4], [5], [18], [19]. $AC@k$ represents the accuracy that top k results include the root causes related KPIs for all anomaly cases. A higher $AC@k$ score, especially for small values of k, indicates the algorithm correctly localizes the actual root cause. Given the anomaly cases set A, $AC@k$ is calculated as follows:

$$AC@k = \frac{1}{|A|} \sum_{a \in A} \frac{\sum_{i<k} R^a[i] \in V_{rc}^a}{min(k, |V_{rc}^a|)},$$

where $R^a[i]$ is the rank of KPIs for anomaly case a. V_{rc}^a is the root cause related KPI set for anomaly case a. $|A|$ represents the number of elements in the set A. The overall

TABLE I: All KPIs used in our research. These KPIs are standard KPIs of MySQL database, Docker and Linux OS.

Type		KPIs
Workload (12)	Com (8)	mysql.qps; mysql.tps; mysql.insert_ps; mysql.update_ps; mysql.delete_ps; mysql.commit_ps; mysql.insert_select_ps; mysql.replace_select_ps; mysql.replace_ps
	Rows (4)	mysql.innodb_rows_inserted; mysql.innodb_rows_read; mysql.innodb_rows_deleted; mysql.innodb_rows_updated
Storage Engine IO (18)	Physical IO (10)	mysql.bytes_received; mysql.bytes_sent; mysql.innodb_data_read; mysql.innodb_data_reads;mysql.innodb_data_writes; mysql.innodb_data_written; mysql.io_bytes_read; mysql.io_bytes_write; mysql.io_bytes;mysql.innodb_data_written
	Logic IO (8)	mysql.innodb_buffer_pool_reads_requests; mysql.innodb_bp_usage_pct; mysql.Innodb_buffer_pool_pages_flushed; mysql.innodb_buffer_pool_reads; mysql.innodb_buffer_pool_write_requests; mysql.innodb_bp_data_mbytes; mysql.innodb_log_writes; mysql.innodb_data_fsyncs
Instance Resource (16)	CPU (2)	mysql.cpu_usage; docker.cpu_usage;
	Memory (5)	mysql.mem_used; docker.mem_uasge_percent; docker.mem_used; docker.mem_cache; docker.mem_buffer
	Network Card (2)	docker.net_send; docker.net_recv
	Disk IO (2)	docker.io_read; docker.io_write
	Storage (2)	mysql.storage_data; mysql.storage_log
	Session (3)	mysql.active_session; mysql.threads_connected; mysql.total_session
Host Resource (33)	CPU (8)	cpu.hi_usage; cpu.usage; cpu.usage_percent; cpu.user_usage; cpu.sys_usage; cpu.si_usage; cpu.iowait_usage; minion.cpu_usage
	Memory (10)	mem.usage; mem.used; mem.swap_used; mem.cached; mem.buffers; mem.pgpgout; mem.pgpgin; mem.cache; mem.buffer; minion.mem_used
	Network Card (4)	net.recv; net.send; net.recv_usage; net.send_usage
	Disk IO (11)	disk.reads; disk.writes; disk.rkB_ps; disk.wkB_ps; disk.read_rt; disk.write_rt; disk.queue; disk.io_rt; disk.iops; disk.svctm; disk.util
Business (3)	Response Time (3)	rt; rt_avg; max_rt

TABLE II: Details of injected 150 database anomalies for evaluation.

Type of anomaly	Number	Description
CPU Saturation	30	Invoke stress-ng, which starts N workers that perform various matrix operations on floating point values.
Network Congestion	30	Simulate network congestion by adding an artificial 500-milliseconds delay to every traffic over the network via Linux's tc (Traffic Control) command.
IO Saturation	30	Invoke stress-ng, which starts N workers that perform a mix of sequential, random and memory mapped read/write operations as well as forced sync'ing and cache dropping.
Memory Saturation	30	Invoke stress-ng, which starts N workers that grow their heaps by reallocating memory.
Anomalous workload	30	Greatly increase the rate of transactions and the number of clients simulated by OLTPBenchmark (150 additional terminals with transaction rate of 50,000).

performance of each algorithm is evaluated by computing the average $Avg@k$:

$$Avg@k = \frac{1}{k} \sum_{1 \leq j \leq k} AC@j.$$

D. Baseline Comparison

FluxInfer was evaluated against nine baseline approaches, which also localize the root cause by analyzing KPIs: PAL [20], CauseInfer [3], CloudRanger [4], MS-Rank [7], Microscope [6], MicroRCA [21], MonitorRank [18], TON18 [19], and MicroCause [5]. Table III summarizes the approaches of dependency relationship learning and root cause inference of these works. The details of them are introduced in the related work section (§VI).

Table III shows the evaluation results. The AC@3, AC@5, and Avg@5 of FluxInfer are 0.9, 0.95, and 0.77, outperforming the other nine baselines. Due to that the database anomalous KPIs almost change together, it is difficult to confirm which data point is the starting point of the anomalous change. Further, the noises and fluctuations could affect the change start time detection. Therefore, the root cause related KPIs cannot be directly localized by the start times of anomalous changes. This is the reason why PAL [20] cannot achieve good performance. Due to the unknown latent variables that are hardly observed and inferred, there are some incorrect dependency relationships in the directed acyclic graphs constructed by PC-algorithm [8]. Then because of traversing the inaccurate dependency graphs, CauseInfer [3], CloudRanger [4], Microscope [6], MicroRCA [21], MonitorRank [18], and TON18 [19] cannot achieve good performance. It is also observed that the performance of MicroCause is better than other PC-algorithm based baselines. The reason is that MicroCause used pre-defined rules for root cause localization. These rules are more suitable for diagnosing the type of anomalous workload database anomaly (Table II). Therefore, **the performances of these PC-algorithm based baselines demonstrate the value of our weighted undirected dependency graph design**.

It is essential to highlight that these root cause inference algorithms (Deep First Search, Second-order Random Walk, Traversing+Pearson Correlation, Personalized PageRank, Random Walk, and TCORW) are all designed for the *directed* acyclic dependency graph, which can not be applied in the *undirected* dependency graph (*e.g.*, WUDG) of FluxInfer.

E. Robust Anomaly detection

To evaluate the design of the robust anomaly detection algorithm, we designed two comparison algorithms: FluxInfer-with-CUSUM and FluxInfer-without-AD. In FluxInfer-with-CUSUM, we replaced the robust anomaly detection algorithm of FluxInfer with the CUSUM [22] algorithm, a commonly used robust anomaly detection algorithm. In FluxInfer-

TABLE III: The evaluation results of different algorithms

Algorithm	Relationships Learning	Root Cause Inference	AC@1	AC@2	AC@3	AC@5	Avg@5
PAL [20]	N/A	Anomaly Time Order	0.09	0.12	0.14	0.20	0.14
CauseInfer [3]	PC Algorithm	Deep First Search	0.12	0.20	0.22	0.28	0.21
CloudRanger [4], MS-Rank [7]	PC Algorithm	Second-order Random Walk	0.08	0.19	0.27	0.36	0.24
Microscope [6]	PC Algorithm	Traversing+Pearson Correlation	0.06	0.11	0.16	0.24	0.15
MicroRCA [21]	PC Algorithm	Personalized PageRank	0.08	0.17	0.30	0.38	0.25
MonitorRank [18], TON18 [19]	PC Algorithm	Random Walk	0.08	0.16	0.28	0.39	0.24
MicroCause [5]	PCTS	TCORW	0.23	0.38	0.47	0.60	0.44
FluxInfer	**WUDG**	**Weighted PageRank**	**0.43**	**0.69**	**0.90**	**0.95**	**0.77**
FluxInfer-with-CUSUM	WUDG	Weighted PageRank	0.23	0.40	0.62	0.73	0.53
FluxInfer-without-AD	WUDG	Weighted PageRank	0.13	0.20	0.32	0.50	0.38

without-AD, we removed the robust anomaly detection algorithm of FluxInfer. All KPIs (normal KPIs and anomalous KPIs) are used to construct the dependency graph and localize the root cause. Table III shows the evaluation results. Due to that the essence of CUSUM is the cumulative sum, the irregular noises and fluctuations in KPIs will affect the cumulative sum. Then the anomalous changes cannot be detected effectively. Therefore, FluxInfer-with-CUSUM cannot achieve good performance. FluxInfer-without-AD uses normal KPIs and anomalous KPIs to construct the dependency graph and localize the root cause. However, the dependency graph contains many dependencies among normal KPIs. FluxInfer-without-AD could recommend some root cause *unrelated* normal KPIs, which leads to poor performance. Therefore, **the results of FluxInfer-with-CUSUM and FluxInfer-without-AD demonstrate the effectiveness of our robust anomaly detection design**. Further, we can see that the performance of FluxInfer-with-CUSUM is better than that of FluxInfer-without-AD, which demonstrates that **anomaly detection is necessary for the design of FluxInfer**.

F. Diagnosis Time

Diagnosing online anomalies must be as quickly as possible because the diagnosis time is always related to the duration of the recovery time. Therefore, the running time of FluxInfer should be as shorter as possible so that operators can quickly take mitigation measures. The running time of FluxInfer consists of three parts: anomaly detection, dependency graph construction, and root cause localization. We used ten processes of our server to run anomaly detection for KPIs of each case concurrently. After evaluation, the average running time of FluxInfer is 53 seconds. We interviewed five experienced database operators from two cloud service providers. They all confirmed that the average online diagnosis time must be less than one minutes. Therefore, the diagnosis time of FluxInfer can satisfy the online diagnosis.

G. Case Study

CPU saturation is a type of common online database anomaly. Figure 7 shows the diagnosis result of FluxInfer and MicroCause [5] for a CPU saturation case. There are 71 anomalous KPIs in this case, and the root cause related KPIs are cpu.usage and cpu.usage_percent. The diagnosis result of FluxInfer shows that the root cause related KPIs are ranked top 1 and top 2, respectively. However, the root cause related KPI is ranked top 7 in the diagnosis result of MicroCause, and the other eight baselines even cannot rank the root cause related KPIs in top 10. In this scenario, the CPU saturation

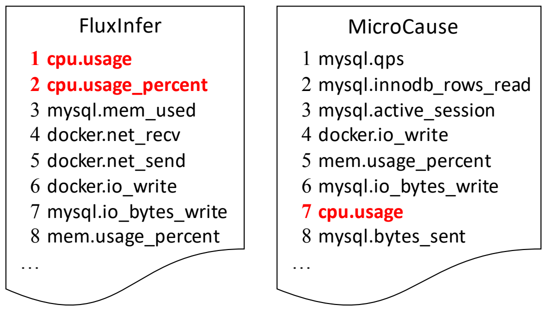

Fig. 7: The diagnosis results of FluxInfer and MicroCause [5] for a CPU saturation case. The bold red font represents the root cause related KPI.

anomaly caused the workload KPIs to drop suddenly. Due to that MicroCause gives the workload KPIs higher priority to be considered as the root cause, the workload KPIs (mysql.qps and mysql.innodb_rows_read) are ranked top 1 and top 2. The diagnosis result of MicroCause demonstrates that the root cause inference rules designed for microservices are not applicable in the database system. Further, the DAG's incorrect dependency relationships also affect the ranking of the root cause related KPIs.

VI. Related Work

It is vitally important to quickly localize the root causes of system failures for assuring the quality of users' experience. Many solutions have been proposed to localize root causes KPIs in databases, clouds, and microservices. Next, we will introduce the details of these works.

Dependency graph based methods: CloudRanger [4], Microscope [6], MS-Rank [7], and MicroCause [5] localized the root cause of anomalies in clouds or microservices by automatically constructing the dependency graphs among KPIs. They used PC-algorithm [8] to construct the directed acyclic dependency graphs. Then, they applied different algorithms to traverse the graph for root cause localization. CauseInfer used DFS (Depth First Search) algorithm, CloudRanger and MS-Rank used a second-order random walk algorithm. Microscope traversed the graph by some defined rules to obtain root cause candidates. These candidates were then ranked by the Pearson correlation coefficients between the front end and the abnormal service instances. MicroCause designed a Path Condition Time Series (PCTS) algorithm (an improved algorithm based on PC algorithm) to learn the dependency graph of KPIs, and a Temporal Cause Oriented Random Walk (TCORW) approach to localize the root cause related KPIs.

MonitorRank [18], TON18 [19], and MicroRCA [21] also focused on the anomaly diagnosis of clouds or microservices. They used the pre-defined calling typologies to construct the dependency graphs among KPIs. MonitorRank constructed the call graph of the system's APIs (Application Programming Interface), which was easily obtained by batch processing systems (*e.g.*, Hadoop), to learn APIs' dependencies. TON18 used the APIs of OpenStack to obtain physical dependencies, and a popular trace analysis tool, PreciseTracer, to capture call dependencies. MicroRCA constructed an attributed graph to represent the dependencies in microservices environments. After obtaining the dependencies, MonitorRank and TON18 applied a random walk algorithm for root cause localization. MicroRCA used the Personalized PageRank algorithm for root cause localization.

Other methods: DBSherlock [2] explains the anomaly in the form of predicates and possible causes produced by causal models. These explanations only *assist* operators in diagnosing anomalies. The models need operators to explain by their domain knowledge and experience manually, and the root cause localization process needs the feedback of operators. PerfXplain [23] used decision trees to explain the performance of map-reduce jobs automatically. PerfAugur [24] used robust statistics for automatic performance explanation of cloud services. However, these approaches are more likely to find secondary symptoms when the root cause of the anomaly is outside the database and not directly captured by the collected KPIs. Scorpion [25] used sensitivity-analysis-based techniques to find the individual tuples most responsible for extreme aggregate values in scientific computations, which are not applicable for database anomaly diagnosis. The reason is that databases often avoid prohibitive data collecting overheads by maintaining aggregate statistics rather than detailed statistics for individual transactions. Works [26], [27] proposed tools to an automatic diagnosis of commercial database anomalies. However, work [26] needs DBAs to provide a set of manual rules, and work [27] needs detailed internal performance measurements from the database system. PAL [20] localized the root cause by sorting the change start times of KPIs' abnormal changes, which supposes that the root cause related KPIs change firstly.

VII. Conclusions

To relieve database operators from the tedious and time-consuming manual diagnosis work, we propose FluxInfer, which can automatically and accurately localize the root cause related KPIs of online database performance anomalies. Our evaluation on testbed shows that the AC@3, AC@5, and Avg@5 of FluxInfer are 0.90, 0.95, and 0.77, outperforming nine baselines by 64%, 60%, and 53% on average. The diagnosis time of FluxInfer can satisfy the online diagnosis.

Acknowledgment

The authors thank Ruru Zhang for the helpful work on baseline comparison. This work has been supported by the National Key R&D Program of China (2019YFB1802504), the National Natural Science Foundation of China (61902200), the China Postdoctoral Science Foundation (2019M651015), the Key-Area Research and Development Program of Guangdong Province (2019B010136001), the Science and Technology Planning Project of Guangdong Province LZC0023, and the Beijing National Research Center for Information Science and Technology (BNRist) key projects.

References

[1] M. Ma, Z. Yin, S. Zhang *et al.*, "Diagnosing root causes of intermittent slow queries in cloud databases," *Proceedings of the VLDB Endowment*, vol. 13, no. 8, pp. 1176–1189, 2020.

[2] D. Y. Yoon, N. Niu *et al.*, "Dbsherlock: A performance diagnostic tool for transactional databases," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 1599–1614.

[3] P. Chen, Y. Qi, P. Zheng, and D. Hou, "Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 1887–1895.

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

[5] Y. S. Yuan Meng, Shenglin Zhang *et al.*, "Localizing failure root causes in a microservice through causality inference," in *IWQoS 2020*.

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

[7] M. Ma, W. Lin *et al.*, "Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications," in *ICWS 2019*.

[8] M. Kalisch and P. Bühlmann, "Estimating high-dimensional directed acyclic graphs with the pc-algorithm," *Journal of Machine Learning Research*, vol. 8, no. Mar, pp. 613–636, 2007.

[9] W. Xing and A. Ghorbani, "Weighted pagerank algorithm," in *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004*. IEEE, 2004, pp. 305–314.

[10] L. G. Neuberg, "Causality: Models, reasoning, and inference , by judea pearl, cambridge university press, 2000," *Econometric Theory*, vol. 19, no. 04, pp. 675–685, 2003.

[11] D. Reynolds, "Gaussian mixture models," *Encyclopedia of biometrics*, pp. 827–832, 2015.

[12] S. Kobayashi, K. Otomo, K. Fukuda, and H. Esaki, "Mining causality of network events in log data," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 53–67, 2017.

[13] R. E. Neapolitan *et al.*, *Learning bayesian networks*. Pearson Prentice Hall Upper Saddle River, NJ, 2004, vol. 38.

[14] E. Yan and Y. Ding, "Discovering author impact: A pagerank perspective," *Information processing & management*, 2011.

[15] D. E. Difallah, A. Pavlo *et al.*, "Oltp-bench: an extensible testbed for benchmarking relational databases," in *VLDB 2013*.

[16] "Stress-ng." https://kernel.ubuntu.com/ cking/stress-ng/.

[17] "Tpc-c." http://www.tpc.org/tpcc/.

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

[19] J. Weng, J. H. Wang, J. Yang, and Y. Yang, "Root cause analysis of anomalies of multitier services in public clouds," *IEEE ACM Transactions on Networking*, vol. 26, no. 4, pp. 1646–1659, 2018.

[20] H. Nguyen, Y. Tan, and X. Gu, "Pal: P ropagation-aware a nomaly l ocalization for cloud hosted distributed applications," in *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*. ACM, 2011, p. 1.

[21] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Microrca: Root cause localization of performance issues in microservices," in *NOMA 2020*.

[22] F. Gustafsson and F. Gustafsson, *Adaptive filtering and change detection*. Citeseer, 2000, vol. 1.

[23] N. Khoussainova, M. Balazinska, and D. Suciu, "Perfxplain: debugging mapreduce job performance," *Proceedings of the VLDB Endowment*, vol. 5, no. 7, pp. 598–609, 2012.

[24] S. Roy, A. C. König *et al.*, "Perfaugur: Robust diagnostics for performance anomalies in cloud services," in *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 2015, pp. 1167–1178.

[25] H. L. Stahnke, "Scorpion nomenclature and mensuration," *Entomological news*, vol. 81, no. 12, pp. 297–316, 1970.

[26] D. G. Benoit, "Automatic diagnosis of performance problems in database management systems," in *Second International Conference on Autonomic Computing (ICAC'05)*. IEEE, 2005, pp. 326–327.

[27] K. Dias, M. Ramacher *et al.*, "Automatic performance diagnosis and tuning in oracle." in *CIDR*, 2005, pp. 84–94.