ABSTRACT

With the growing market of cloud databases, careful detection and elimination of slow queries are of great importance to service stability. Previous studies focus on optimizing the slow queries that result from internal reasons (e.g., poorly-written SQLs). In this work, we discover a different set of slow queries which might be more hazardous to database users than other slow queries. We name such queries Intermittent Slow Queries (iSQs), because they usually result from intermittent performance issues that are external (e.g., at database or machine levels). Diagnosing root causes of iSQs is a tough but very valuable task.

This paper presents iSQUAD, Intermittent Slow Query Anomaly Diagonoser, a framework that can diagnose the root causes of iSQs with a loose requirement for human intervention. Due to the complexity of this issue, a machine learning approach comes to light naturally to draw the interconnection between iSQs and root causes, but it faces challenges in terms of versatility, labeling overhead and interpretability. To tackle these challenges, we design four components, i.e., Anomaly Extraction, Dependency Cleansing, Type-Oriented Pattern Integration Clustering (TOPIC) and Bayesian Case Model. iSQUAD consists of an offline clustering & explanation stage and an online root cause diagnosis & update stage. DBAs need to label each iSQ cluster only once at the offline stage unless a new type of iSQs emerges at the online stage. Our evaluations on real-world datasets from Alibaba OLTP Database show that iSQUAD achieves an iSQ root cause diagnosis average F1-score of 80.4%, and outperforms existing diagnostic tools in terms of accuracy and efficiency.

PVLDB Reference Format:

1Work was done while the author was interning at Alibaba Group.
2Work was done while the author was a visiting scholar at Alibaba Group.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publications rights licensed to the VLDB Endowment.

1. INTRODUCTION

The growing cloud database services, such as Amazon Relational Database Service, Azure SQL Database, Google Cloud SQL and Alibaba OLTP Database, are critical infrastructures that support daily operations and businesses of enterprises. Service interruptions or performance hiccups in databases can lead to severe revenue loss and brand damage. Therefore, databases are always under constant monitoring, where the detection and elimination of slow queries are of great importance to service stability. Most database systems, such as MySQL, Oracle, SQL Server, automatically log detailed information of those queries whose completion time is over a user-defined threshold [7][37][43], i.e., slow queries. Some slow queries result from internal reasons, such as nature of complexity, lack of indexes and poorly-written SQL statements, which can be automatically analyzed and optimized [13][32][44]. Many other slow queries, however, result from intermittent performance issues that are external (e.g., at database or machine levels), and we name them Intermittent Slow Queries (iSQs).

Usually, iSQs are the cardinal symptom of performance issues or even failures in cloud databases. As iSQs are intermittent, service developers and customers expect them to be responsive as normal, where sudden increases of latency have huge impacts. For example, during web browsing, an iSQ may lead to unexpected web page loading delay. It has been reported that every 0.1s of loading delay would cost Amazon 1% in sales, and every 0.5s of additional load delay for Google search results would lead to a 20% drop in traffic [30]. We obtain several performance issue records carefully noted by DBAs of Alibaba OLTP Database in a year span: when a performance issue occurs, a burst of iSQs lasts for minutes. As a matter of fact, manually diagnosing root causes of iSQs takes tens of minutes, which is both time consuming and error-prone.

Diagnosing root causes of iSQs gets crucial and challenging in cloud. First, iSQ occurrences become increasingly common. Multiple database instances may reside on the same physical machines for better utilization, which in turn can cause inter-database resource contentions. Second, root causes of iSQs vary greatly. Infrastructures of cloud databases are more complex than those of on-premise databases [29], making it harder for DBAs to diagnose root causes. Precisely, this complexity can be triggered by instance migrations, expansions, storage decoupling, etc. Third, massive database instances in cloud make iSQs great in population. For example, tens of thousands of iSQs are generated in Alibaba OLTP Database per day. In addition, roughly 83% of enterprise workloads are forecasted to be in the cloud by 2020 [12]. This trend makes it critical to efficiently diagnose the root causes of iSQs.

In this work, we aim to diagnose root causes of iSQs in cloud databases with minimal human intervention. We learn about symp-
toms and root causes from failure records noted by DBAs of Alibaba OLTP Database, and we underscore four observations:

1) **DBAs need to scan hundreds of Key Performance Indicators (KPIs) to find out performance issue symptoms.** These KPIs are classified by DBAs to eight types corresponding to different root causes (as summarized in Table 1). Traditional root cause analysis (RCA) [2][3][18], however, does not have the capability of specifically distinguishing multiple types of KPI symptoms to diagnose the root causes of iSQs. For instance, by using system monitoring data, i.e., single KPI alone (or a single type of KPIs), we usually cannot pinpoint iSQs’ root causes [10].

2) **Performance issue symptoms mainly include different patterns of KPIs.** We summarize three sets of symmetric KPI patterns, i.e., spike up or down, level shift up or down, and void. We observe that even if two iSQs have the identical set of anomalous KPIs (but with distinct anomaly behaviors), their root causes can differ. Thus, purely based on detecting KPI anomalies as normal or abnormal we cannot precisely diagnose iSQs’ root causes [6,45].

3) **One anomalous KPI is usually accompanied by another or more anomalous KPIs.** Certain KPIs are highly correlated [24], and rapid fault propagation in databases renders them anomalous almost simultaneously. We observe that the way in which a KPI anomaly propagates can be either unidirectional or bidirectional.

4) **Similar symptoms are correlated to the same root cause.** In each category of root causes, KPI symptoms of performance issues are similar to each other’s. For instance, KPIs in the same type can substitute each other, but their anomaly categories remain constant. Nevertheless, it is infeasible to enumerate and verify all possible causalities between anomalous KPIs and root causes [30].

As a result, iSQs with various KPI fluctuation patterns appear to have complex relationships with diverse root causes. To discover and untangle such relationships, we have made efforts to explore machine learning (ML) based approaches, but have encountered many challenges during this process. First, anomalous KPIs need to be properly detected when an iSQ occurs. Traditional anomaly detection methods recognize only anomalies themselves, but not anomaly types (i.e., KPI fluctuation changes such as spike up or down, level shift up or down). The availability of such information is vital to ensure high accuracy of subsequent diagnoses. Second, based on detected KPI fluctuation patterns, the root cause of that iSQ has to be identified from numbers of candidates. Standard supervised learning methods are not suitable for such diagnoses because the case-by-case labeling of root causes is prohibitive. An iSQ can trigger many anomalous KPIs and lead to tremendous investigation, taking hours of DBAs’ labor. Third, though unsupervised learning (e.g., clustering) is an eligible approach to easing the labeling task for DBAs, it only retains limited efficacy to inspect every cluster. It is known to be hard to make clusters that are both intuitive (or interpretable) to DBAs and accurate [25].

To address the aforementioned challenges, we design **iSQUAD** (Intermittent Slow Query Anomaly Diagnoser), a comprehensive framework for iSQ root cause diagnoses with a loose requirement for human intervention. In detail, we adopt **Anomaly Extraction and Dependency Cleansing** in place of traditional anomaly detection approaches to tackle the first challenge of anomaly diversity. For labeling overhead reduction, **Type-Oriented Pattern Integration Clustering (TOPIC)** is proposed to cluster iSQs of the same root causes together, considering both KPIs and anomaly types. In this way, DBAs only need to explore one representative root cause in each cluster rather than label numbers of them individually. For clustering interpretability, we take advantage of **Bayesian Case Model** to extract a case-based representation for each cluster, which is easier for DBAs to investigate. In a nutshell, iSQUAD consists of two stages: an **offline clustering & explanation stage** and an **online root cause diagnosis & update stage**. The offline stage is run first to obtain the clusters and root causes, which are then used by the online stage for future diagnoses. DBAs only need to label each iSQ cluster once, unless a new type of iSQs emerges. By using iSQUAD, we significantly reduce the burden of iSQ root cause diagnoses for DBAs on cloud database platforms.

The key contributions of our work are as follows:

- We identify the problem of Intermittent Slow Queries in cloud databases, and design a scalable framework called iSQUAD that provides accurate and efficient root cause diagnosis of iSQs. It adopts machine learning techniques, while overcoming the inherent obstacles in terms of versatility, labeling overhead and interpretability.
- We apply Anomaly Extraction of KPIs in place of anomaly detection to distinguish anomaly types. A novel clustering algorithm TOPIC is proposed to reduce the labeling overheads.
- To the best of our knowledge, we are the first to apply and integrate case-based reasoning via the Bayesian Case Model [23] in database domain and to introduce the case-subspace representations to DBAs for labeling.
- We conduct extensive experiments for iSQUAD’s evaluation and demonstrate that our method achieves an average F1-score of 80.4%, i.e., 49.2% higher than that of the previous technique. Furthermore, we have deployed a prototype of iSQUAD in a real-world cloud database service. iSQUAD helps DBAs diagnose all ten root causes of several hundred iSQs in 80 minutes, which is approximately thirty times faster than traditional case-by-case diagnosis.

The rest of this paper is organized as follows: §2 describes iSQs, the motivation and challenges of their root cause diagnoses. §3 overviews our framework, iSQUAD [55], discusses detailed ML techniques in iSQUAD that build comprehensive clustering models. §5 shows our experimental results. §6 presents a case study in a real-world cloud database service. §7 reviews the related work, and §8 concludes the paper.

2. **BACKGROUND AND MOTIVATION**

In this section, we first introduce background on iSQs. Then, we conduct an empirical study from database performance issue records to gain some insights. Finally, we present three key challenges in diagnosing the root causes of iSQs.

2.1 **Background**

**Alibaba OLTP Database.** Alibaba OLTP Database (in short as Alibaba Database) is a multi-tenant DBPaaS supporting a number of first-party services including Taobao (customer-to-customer online retail service), Tmall (business-to-consumer online retail service), DingTalk (enterprise collaboration service), Cainiao (logistics service), etc. This database houses over one hundred thousand actively running instances across tens of geographical regions. To monitor the compliance with SLAs (Service-Level Agreements), the database is equipped with a measurement system [39] that continuously collects logs and KPIs (Key Performance Indicators).

**Intermittent Slow Queries (iSQs).** Most database systems, such as MySQL, Oracle, SQL Server, automatically record query time of each query execution [37][43]. The query time is the time between when an SQL query is submitted to, and when its results are returned by, the database. We formally define Intermittent Slow Queries (iSQs) as follows. For a SQL query \( Q \), its \( t \)-th occurrence \( Q_t \) (whose observed execution time is \( T_t \)) is an iSQ if and only if \( X_t > z \) and \( P(X_t > z) < \epsilon \), where \( 1 \leq t \leq T \) is the total
number of $Q$’s recent occurrences), $z$ is slow query threshold, and $\epsilon$ is iSQ probability threshold. For interactive transactional workloads on Alibaba Database, DBAs empirically set $z = 1s$, $\epsilon = 0.01$, and $T = 10^5$. Note that these thresholds can be dynamically tuned (e.g., using percentiles and standard deviations) as workload changes, which however is not the focus of this work. The iSQs occur intermittently, which is guaranteed by the probability threshold $\epsilon$. For example, Fig. 2(a) shows the query time probability distribution of one SQL. In this plot, those queries whose query time is over one second take up 0.0028. These iSQs are resulted from intermittent external performance issues (e.g., at database or machine levels). On the contrary, Fig. 2(b) shows another SQL that is a typical slow query, because it is slow for each execution.

The iSQs account for 1% of the slow queries, but they have a huge impact. Other type of slow queries are mostly caused by the nature of complexity of tasks and are usually non-interactive and tolerable (which takes up about 79%). We already have methods to handle the remaining slow queries (20%) by, e.g., adding indexes or re-writing SQL statements. Though iSQs are small in population, they are still tens of thousands in number every day. Dealing with iSQs is of great importance, since, when they occur unexpectedly, the experience of end users is severely impacted. Therefore, it is critical to design a solution to diagnosing the root causes of iSQs.

2.2 Observations

We obtain several performance issue records from Alibaba OLTP Database in a year span. These records, containing performance issue symptoms and root causes, are recorded by DBAs once performance issues happen. We observe that all records share a common symptom, *i.e.*, a burst of iSQs which last for minutes. When a performance issue occurs, a number of normal queries of online services are affected and become much slower than usual. Thus, understanding the root cause of iSQs is important to mitigate them. Studying the records offers insights to design a root cause analysis framework. In this work, we only consider records that have been resolved. For confidential reasons, we have to hide details of these records and report relatively rough data instead.

**KPIs are important to locate iSQs’ root causes.** When a performance issue arises, DBAs need to scan hundreds of Key Performance Indicators (KPIs) to find its symptoms. A KPI captures a system unit’s real-time performance or behavior in a database system. KPIs are one of the most important and useful monitoring data for DBAs to diagnose performance issues. For example, TCP Response Time (tcp-rt) is used in [9] to detect performance anomalies. Any single KPI alone, however, cannot capture all types of performance issues [35]. Indeed, diverse types of KPIs are tracking various aspects of system status. For instance, usually hundreds of KPIs are monitored for MySQL [3].

In this work, we focus on iSQs’ root causes that can be explained or reflected by KPIs. These KPIs are not only collected from physical machines and docker instances, but also from MySQL configurations. For each iSQ, we obtain the exact time and the location (the instance or physical machine) of the performance issue. With the help of experienced DBAs, we choose 59 KPIs, classified into eight types as shown in Table 1. They cover almost all conceivable features of performance issues that may cause iSQs.

The anomaly types of KPIs should be paid attention to. Performance issue symptoms can be represented by different types of KPI patterns. From these records, KPI symptoms can be summarized into four anomaly types, *i.e.*, spike, level shift-up, level shift-down (KPI witnesses a sudden increase / decrease or ramp-ups / downs for a long time) and void (KPI value is zero or missing), as shown in Fig. 3. Previous anomaly detection algorithms [31, 33] focus on whether KPIs are anomalous or not. However, DBAs not only check the presence of an anomaly, but also pay more attention to the exact type of it.

We present two typical cases where iSQs can occur. Consider the first case in Fig. 4, where two instances (usually without allocating fixed I/O resources in practice) are simultaneously running on the same physical machine. The first instance undertakes a database backup that is unavoidably resource-demanding, and it consequently triggers an I/O related anomaly (reflected in one or more CPU-related KPIs). Since these two instances are sharing a fixed amount of I/O resources, the queries inside Instance 2 are heavily impacted and hence appear to be iSQs. This case suggests that iSQs may occur due to the negative influence of their surrounding environments, such as related or “neighboring” slow queries. The second case involves a physical machine with only one instance on it. If there is a sudden increase in the overall workload of this instance (*e.g.*, caused by an online flash sale event), one or more CPU-related KPIs can become alarmingly anomalous. Hence, queries inside this only instance become iSQs. The second case shows that abnormal workloads may lead to iSQs as well.

**KPI anomalies are highly correlated.** One anomalous KPI may be most of the time accompanied by another one or more anomalous KPIs. Since systems have complex relationships among components, KPIs are highly correlated with each other [24]. We find that fault propagation can be either unidirectional or bidirectional and the relation between two KPIs is not necessarily mu-

![Figure 1: The architecture of the data collection system for Alibaba OLTP Database.](image)

![Figure 2: Query time probability distribution of two SQLs. (a) The long tail part represents the iSQs. (b) Slow queries.](image)

Table 1: KPI types w.r.t instances and physical machines

<table>
<thead>
<tr>
<th>Type</th>
<th># KPIs</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2</td>
<td>dockers.cpu-usage</td>
</tr>
<tr>
<td>I/O</td>
<td>15</td>
<td>mysql.io-bytes</td>
</tr>
<tr>
<td>Workload</td>
<td>13</td>
<td>mysql.tps</td>
</tr>
<tr>
<td>TCP RT</td>
<td>12</td>
<td>tcp-rt99</td>
</tr>
<tr>
<td>Memory</td>
<td>3</td>
<td>mysql.buffer-pool-reads</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical Machine</th>
<th>Type</th>
<th># KPIs</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>6</td>
<td></td>
<td>cpu.usage</td>
</tr>
<tr>
<td>I/O</td>
<td>4</td>
<td></td>
<td>io.wait-usage</td>
</tr>
<tr>
<td>Network</td>
<td>4</td>
<td></td>
<td>net.receive-usage</td>
</tr>
</tbody>
</table>
2.3 Challenges

We encounter three challenges when applying machine learning techniques to our diagnostic framework.

**Anomaly Diversity.** A large number of state-of-the-art anomaly detectors are running, and scrutinizing KPI data all the time. Most of them can quickly tell whether an anomaly occurs, but this type of binary information is not sufficient in our scenario. This is because iSQs tend to simultaneously lead to multiple anomalous KPIs, but in fact the timelines of these KPIs can differ significantly.

Under this special circumstance, distinguishing only between the normal and the abnormal might not produce satisfactory results, again, taking Fig. 4 as an example. They both contain the same seven KPIs but in different anomaly types. We may come to the incorrect conclusion that the two groups of performance issue (iSQs) have the same root causes (while they actually do not). Furthermore, it is preferable to have a method that can achieve high accuracy, low running time, and high scalability in detecting anomalies in large datasets.

**Limitation of existing solutions:** combinations of anomaly types may correspond to various root causes, so current anomaly detectors generally overlook and over-generalize the types of anomalies. Such detectors may erroneously filter out a considerable amount of information in the (monitoring data) pre-processing phase, and thus degrade the quality of the (monitoring) dataset.

**Labeling Overheads.** Suspecting there is strong correspondences and correlations among KPIs’ anomalous performances and their root causes [6, 45], we seek to ascertain such relationships by integrating DBAs’ domain knowledge into our machine learning approaches. To this end, we ask experienced DBAs to label root causes of iSQs. The amount of work, however, is massive if the historical iSQs have to be manually diagnosed case by case.

Even though DBAs have domain knowledge, the labeling process is still painful [31]. For each anomaly diagnosis, a DBA must first locate and log onto a physical machine, and then inspect logs and KPIs related to KPIs to reach a diagnostic conclusion. To successfully do so, DBAs need to understand KPI functionalities & categories, figure out the connections between the anomalous KPIs, comprehend KPI combinations, locate multiple anomalous KPIs & machines & instances, and anticipate possible results & impacts on the quality of services. Typically, DBAs analyze anomalies case by case, but this way of diagnosing them is both time-consuming and labor-intensive. For example, one tricky anomaly diagnosis case handled by an experienced DBA can take hours or even a whole day. Thus, scrutinizing raw data is tedious and error-prone, whereas the error tolerance level we can afford is very low.

**Interpretable Models.** Being able to explain or narrate what causes the problem when it arises (which we call the interpretability) is essential in our case. To be able to do so, DBAs need to be presented with concrete evidence of subpar machine and instance per-
formances, such as anomalous KPIs, so that they can take actions accordingly. DBAs typically do not fully trust in machine learning black-box models for drawing conclusions for them, because those models tend to produce results that are hard to generalize, while real-time analyses have to deal with continuously changing scenarios with various possible inputs. Therefore, we need to design our diagnostic framework for better interpretability.

Unfortunately, an inevitable trade-off exists between a model’s accuracy and its interpretability to human [25]. This issue arises because the increasing system complexity boosts its accuracy at the cost of interpretability, i.e., human can hardly understand the result and the intricacy within the model as it becomes too complicated. Therefore, how to simultaneously achieve both good interpretability and high accuracy in our analysis system and how to push the trade-off frontier outwards are challenging research problems.

**Limitation of existing solutions:** Employing decision trees [15] to explain models is quite common. For example, DBSherlock [45] constructs predicate-based illustrations of anomalies with a decision-tree-like implementation. The reliability, however, depends heavily on feeding precise information at the onset, because even a nuance in input can lead to large tree modifications, which are detrimental to the accuracy. Further, decision trees may also incur the problem of “paralysis of analysis”, where excessive information instead of key elements is presented to decision makers. Excessive information could significantly slow down decision-making processes and affect their efficiencies.

### 3. OVERVIEW

We design a framework – iSQUAD (Intermittent Slow QUery Anomaly Diagnoser), as shown in [Fig. 5](#). The iSQUAD framework consists of two stages: an *offline analysis & explanation* and an *online root cause diagnosis & update*. This design of separation follows the common pattern of offline learning and online applying.

Typically, iSQs with the same or similar KPIs have the same root causes. Thus, it is necessary that the model should draw connections between iSQs and their root causes. DBAs may participate to investigate this connection with high accuracy because of their domain knowledge. It is infeasible to directly assign root causes to iSQ clusters without labeling. Hence, the offline stage is primarily for clustering iSQs based on a standard and presenting them to DBAs who can more easily recognize and label root causes. We feed datasets of past iSQs to the offline stage, and then concentrate on certain intervals given specific timestamps. Things become straightforward as we can focus on only selected time intervals from KPIs’ timelines and undertake anomaly extraction on KPIs within the intervals. Next, we have all anomalous KPIs discretized. Then, we apply the dependency cleansing on this partial result. Namely, if we have two abnormal KPIs A and B, and we have domain knowledge that A’s anomaly tends to trigger that of B, we “cleanse” the anomaly alert on B. Hence, we can assume that all the anomalies are independent after this step. We then perform the Type-Oriented Pattern Integration Clustering (TOPIC) to obtain a number of clusters. For each cluster, we apply the Bayesian Case Model to get a prototypical iSQ and its fundamental KPI anomalies as the feature space to represent this whole cluster. Finally, we present these clusters with their representations to DBAs who investigate and assign root causes to iSQ clusters.

In the *online root cause diagnosis & update* stage, iSQUAD automatically analyzes an incoming iSQ and its KPIs. We execute the online anomaly extraction and dependency cleansing like in the offline stage and gain its abnormal KPIs. Subsequently, we match the query to a cluster. Specifically, we compare this query with every cluster based on the similarity score, and then match this query with the cluster whose pattern is the closest to this query’s. After that, we use the root cause of this cluster noted by DBAs to help explain what triggers this iSQ. If the query is not matched with any existing clusters, a new cluster is generated and DBAs will investigate and assign a root cause to it. New discovery in the online stage can update the offline stage result.

### 4. iSQUAD DETAILED DESIGN

In this section, we introduce the details of iSQUAD, whose components are linked with our observations in [§2.2]. Gaining insights from the first and second observations, we need an Anomaly Extraction approach to extract patterns from KPI statistics at the time of iSQs’ occurrences in order to accurately capture the symptoms [§4.1.1]. According to the third observation, we must eliminate the impact of fault propagation of KPIs. Thus, we design a Dependency Cleansing strategy to guarantee the independence among KPI anomalies [§4.1.2]. Based on the fourth observation, similar symptoms are correlated to the same root causes. Therefore, we propose TOPIC, an approach to clustering queries based on anomaly patterns as well as KPI types [§4.1.3]. Since clustering results are not interpretable enough to identify all root causes due
to the lack of case-specific information, the Bayesian Case Model (BCM) is utilized to extract the “meanings” of clusters §4.1.4.

4.1 Offline Analysis and Explanation

4.1.1 Anomaly Extraction

Given the occurrence timestamps of iSQs, we can collect the related KPI segments from the data warehouse (as shown in Fig. 1). As previously discussed, we need to extract anomalies type from the KPIs. For example, we determine whether a given anomaly is a spike up or down, level shift up or down, even void, corresponding to part (a), (b), (c) (d) in Fig. 3 respectively. We catch this precious information as it can be exceptionally useful for query categorization and interpretation.

To identify spikes, we apply Robust Threshold [9] that suits this situation quite well. As an alternative to the combination of mean and standard deviation to decide a distribution, we use the combination of median and median absolute deviation, which works much more stably because it is less prone to uncertainties like data turbulence. To further offset the effect of data aberrations, the Robust Threshold utilizes a Cauchy distribution in place of the normal distribution, as the former one functions better in case of many outliers. The observation interval is set to one hour by default and the threshold is set empirically.

For level shifts, given a specific timestamp, we split the KPI timeline at that point and generate two windows. Next, we examine whether the distributions of the two timelines are alike or not. If a significant discrepancy is present and discovered by T-Test [39] (an inferential statistic for testing two groups’ mean difference), iSQUAD will determine that a level shift occurs. For level-shift detection, the window is set to 30 minutes by default and the t-value threshold is set empirically.

Note that there are various other excellent anomaly detectors and algorithms, but comparing anomaly detectors is not a contribution of this work. As far as we can tell from our observation, this set of anomaly extraction methods is both accurate and practical.

4.1.2 Dependency Cleansing

To better understand the KPIs’ impacts on iSQs, we need to ensure that all the KPIs chosen for consideration are independent from each other, so that no correlation or over-representation of KPIs impacts our result. To cleanse all potential underlying dependencies, a comparison for each pair of KPIs is necessary. As aforementioned, two KPI anomalies do not necessarily have a mutual correlation. Therefore, unlike some previous works that calculate the mutual information for comparison (e.g., DBSherlock), we apply the confidence [1] based on the association rule learning between two KPIs to determine whether the two KPIs have a correlation. Confidence indicates the number of times the if-then statements are found true.

$$\text{confidence}(A \rightarrow B) = \frac{|A \cap B|}{|A|}$$  (1)

where A and B represent two arbitrary KPIs. Specifically, the confidence from A to B is the number of the co-occurrences of A’s anomalies and B’s anomalies divided by the number of the occurrences of A’s anomalies.

The confidence value spans from 0 to 1, with the left extreme suggesting complete independence of two KPIs and the right extreme complete dependence. In this case, not only 1 denotes dependence. Instead, within the interval, we set a threshold above which two KPIs are considered dependent to reflect real-life scenarios. We permute all KPIs and apply this strategy to each KPI pair.

4.1.3 Type-Oriented Pattern Integration Clustering

To begin with, we familiarize readers with some preliminaries and terminologies used in this section. A pattern encapuslates the specific combination of KPI states (normal or of one of the anomaly categories) for an iSQ. To illustrate, two queries in Fig. 6 have two similar but different patterns. As long as there is one or more discrepancies in between, two patterns are considered different. A KPI type (e.g., CPU-related KPIs, I/O-related KPIs) indicates the type that this KPI belongs to. It comprises one or more KPIs while a KPI falls into one KPI type only. We can roughly categorize KPIs and their functionalities based on KPI types Table 1.

Based on the observations in §2.2, we need to consider both the patterns of iSQs and different types of KPIs to compute the similarity. We define the similarity $S_{ij}$ of two iSQs $i$ and $j$ as follows:

$$S_{ij} = \sqrt{\frac{\sum_{t=1}^{T} |k_{it}, k_{jt}|^2}{T}}$$  (2)

where $t$ is the number of KPI types and $T$ denotes the sum of all $t$’s. $k_{it}$ and $k_{jt}$ are the KPI’s anomaly states in KPI type $t$ of iSQ $i$ and $j$, respectively. The idea behind this definition is to calculate the quadratic mean of the similarity scores with respect to each type of KPIs. Since the quadratic mean is no smaller than the average, it guarantees that minimal KPI changes could not result in grouping the incident with another root cause. $|k_{it}, k_{jt}|$ is the similarity of each KPI type, shown in Equation 3:

$$|k_{it}, k_{jt}| = \frac{\#\text{Matching Anomaly States}}{\#\text{Anomaly States}}$$  (3)

This is the Simple Matching Coefficient [44], which computes two elements’ similarity in a bitwise way. We adopt Simple Matching Coefficient because it reflects how many KPIs possess the same anomaly types. The relatively large number of indicators in certain types of KPIs, however, may dominate compared with other indicators that are minor in population. For instance, imagine that the KPI type “I/O” consists of 18 KPI states while its “CPU” counterpart has only 2 (Table 1). Theoretically, a high score of similarity in “CPU” is prone to be out-weighted by a weak similarity in “I/O”. This “egalitarian” method is not what we expect. To solve this problem, we decide to separate the KPIs based on their types and calculate the individual simple matching coefficient for each KPI type. By doing so, for each KPI type, every pair of iSQs would have a “partial similarity” (opposed to the “complete similarity”)

<table>
<thead>
<tr>
<th>KPI Type</th>
<th>CPU</th>
<th>I/O</th>
<th>Network</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>iSQ1</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>iSQ2</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
</tbody>
</table>

Figure 6: Two queries with various KPIs and similar patterns.

For example, an anomaly in an instance’s CPU utilization usually comes with an anomaly in that of the instance’s physical machine. Therefore, these two KPIs are positively associated to a large extent. If we compute the confidence, we may get the result “1”, which suggests that the two KPIs are dependent. Consequently, we drop all anomalies of physical machine’s CPU utilization and keep those of instance’s CPU utilization. In this part, we cleanse KPI anomalies considering anomaly propagation and reserve the source KPI anomalies. Our rules and results of Dependency Cleansing are verified by experienced DBAs as demonstrated in §5.4.
that we would obtain from taking the quadratic mean of the similarities of all KPIs) with the value in the interval $[0, 1]$.

We describe the details of the clustering procedure as shown in Algorithm 1. The dataset $S$, converted into a dictionary, contains iSQs and their patterns discretized by Anomaly Extraction and Dependency Cleansing. The required input (i.e., threshold $\sigma$) is used to determine how similar two iSQs should be to become homogeneous. To start with, we reverse $S$ into $D$: the indices and values of $D$ are respectively the values (patterns) and clustered indices (iSQs) of $S$ (Line 2 to 3 in Algorithm 1). For the all-zero pattern, i.e., KPI states are all normal, we eliminate it and its corresponding iSQs from $D$ and put them into the cluster dictionary $C$ (Line 4 to 6). This prerequisite checking guarantees that the iSQs with all-zero pattern can be reasonably clustered together. The all-zero pattern does not mean flawless. On the contrary, it usually implies problems with the MySQL core, and it is out of the scope of this paper. Another purpose of this checking is to differentiate the patterns of anomalous DBAs, we conclude that cases and influential indicators are much more intuitive for diagnosis than plain-text statements. More specifically, we expect to spot and select significant and illustrative indicators to represent clusters. To realize this, we take advantage of the Bayesian Case Model (BCM) [23] that is quite suitable for this scenario. BCM is an excellent framework for extracting prototypical cases and generating corresponding feature subspace. With high accuracy preserved, BCM’s case-subspace representation is also straightforward and human-interpretable. Therefore, it is expected to enhance our model’s interpretability by generating and presenting iSQ cases and their patterns for each cluster.

BCM has some specifications that need to be strictly followed. First, it allows only discrete numbers to be present in the feature spaces. According to the original BCM experiment [23], it selects a few concrete features that play an important role in identifying the cluster and the prototypical case. By analogy, we need to use BCM to select several KPIs to support a leading or representative iSQ for each cluster. Originally, the KPI timelines are all continuous data collected directly from the instances or machines, so we discretize them to represent different anomaly types in order to meet this precondition. The discretization is achieved by Anomaly Extraction as discussed in §4.1.3. The second requirement is that labels, i.e., cluster IDs, need to be provided as input. Namely, we need to first cluster the iSQs and then feed them to BCM. Fortunately, we solve this problem with the TOPIC model as discussed in §4.1.3.

In a nutshell, we meet the application requirements of BCM so can apply it to produce the cases and feature subspaces for clusters. With the help of those pieces of information, we are able to understand the result of clusters, and we can thus deliver more suggestive information to DBAs.

### 4.1.4 Bayesian Case Model

With results of TOPIC, we aim to extract useful and suggestive information from each cluster. Based on interviews to eight experienced DBAs, we conclude that cases and influential indicators are much more intuitive for diagnosis than plain-text statements. More specifically, we expect to spot and select significant and illustrative indicators to represent clusters. To realize this, we take advantage of the Bayesian Case Model (BCM) [23] that is quite suitable for this scenario.
In this section, we describe how we initialize and conduct our experiments to assess iSQUAD and its major designs. We evaluate the whole iSQUAD framework, as well as individual components, i.e., Anomaly Extraction, Dependency Cleansing, TOPIC and BCM.

## 5. Setup

### Datasets of Intermittent Slow Queries

We use a large number of real-life iSQs, collected from diverse service workloads of Alibaba OLTP Database, as our datasets in this study. These services are complex online service systems that have been used by millions of users across the globe. All of these diverse services are in various application areas such as online retail, enterprise collaboration and logistics, developed by different groups. We first randomly select application areas such as online retail, enterprise collaboration and logistics, developed by different groups. We first randomly select

### KPIs

We obtain KPIs in a time period from the data warehouse, i.e., the end part of the lower branch in Fig. 1. This time period refers to one hour before and after the timestamp at which an iSQ occurs. We sample KPIs every five seconds and the sampling interval is sufficient to reflect almost all the KPI changes during the time period. In particular, we use 59 KPIs in total, which are carefully selected by DBAs from hundreds of KPIs as the representatives.

#### Ground Truth

We ask DBAs of Alibaba to label the ground truth for evaluating each component of iSQUAD and the overall framework of iSQUAD. For Anomaly Extraction, DBAs carefully label each type of anomaly KPIs. For Dependency Cleansing, DBAs label the aforementioned 59 KPIs and discover ten underlying dependencies among them. For both TOPIC and BCM, DBAs carefully label iSQs' ten root causes.

#### Evaluation Metrics

To sufficiently evaluate the performance of iSQUAD compared with other state-of-the-art tools, we utilize four widely-used metrics in our study, including F1-score, Weighted Average F1-score, Clustering Accuracy and NMI. More details of these metrics are presented as follows.

**F1-score:** the harmonic mean of precision and recall. It is used to evaluate the performance of iSQUAD online diagnoses (§5.2), Anomaly Extraction (§5.3) and Dependency Cleansing (§5.4).

**Weighted Average F1-score** [11]: commonly used in multi-label evaluation. Each type of root causes is a label. We calculate metrics (precision, recall, F1-score) for each label, and find their average weighted by support (the number of true iSQs for each label).

**Clustering Accuracy** [47]: finds the bijective maps between clusters and ground-truth classes, and then measures to what extent each cluster contains the same data as its corresponding class.

**Normalized Mutual Information (NMI)** [8]: a good measure of the clustering quality as it quantifies the “amount of information” obtained about one cluster by observing another cluster.

**Implementation Specifications:** Our framework of iSQUAD is implemented using Python 3.6 and our study is conducted on a Dell R420 server with an Intel Xeon E5-2420 CPU and a 64GB memory.

---

### 4.2 Online Root Cause Diagnosis and Update

By analogy to the offline stage, we follow the same procedures of the Anomaly Extraction and Dependency Cleansing to prepare the data for clustering. After discretizing and cleansing pattern of a new iSQ, iSQUAD can match this query with a cluster for diagnosis. It traverses existing clusters’ patterns to find one pattern that is exactly the same as that of this incoming query, or one that shares the highest similarity score (above the similarity score $\sigma$) with this incoming pattern. If iSQUAD indeed finds one cluster that meets the requirement above, then the root cause of that cluster naturally explains this anomalous query. Otherwise, iSQUAD creates a new cluster for this “founding” query and DBAs are requested to diagnose this query with its primary root cause(s). Finally, the new cluster as well as the diagnosed root cause are added to refine iSQUAD.

When the framework is used to analyze future iSQs, the new cluster, like other clusters, is open for ensuing homogeneous queries if their patterns are similar enough to this cluster’s.

### Table 2: Root causes and corresponding solutions of iSQs labeled by DBAs for the offline clustering (174 iSQs) and online testing dataset (145 iSQs), ordered by the percentage of root causes in the offline dataset.

<table>
<thead>
<tr>
<th>No.</th>
<th>Root Cause</th>
<th>Offline</th>
<th>Online</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Instance CPU Intensive Workload</td>
<td>27.6%</td>
<td>34.5%</td>
<td>Scale up instance CPU</td>
</tr>
<tr>
<td>2</td>
<td>Host I/O Bottleneck</td>
<td>17.2%</td>
<td>17.2%</td>
<td>Scale out host I/O</td>
</tr>
<tr>
<td>3</td>
<td>Instance I/O Intensive Workload</td>
<td>0.9%</td>
<td>15.8%</td>
<td>Scale up instance I/O</td>
</tr>
<tr>
<td>4</td>
<td>Accompanying Slow SQL</td>
<td>8.6%</td>
<td>9.0%</td>
<td>Limit slow queries</td>
</tr>
<tr>
<td>5</td>
<td>Instance CPU &amp; I/O Intensive Workload</td>
<td>8.1%</td>
<td>4.8%</td>
<td>Scale up instance CPU and I/O</td>
</tr>
<tr>
<td>6</td>
<td>Host CPU Bottleneck</td>
<td>7.5%</td>
<td>4.1%</td>
<td>Scale out host CPU</td>
</tr>
<tr>
<td>7</td>
<td>Host Network Bottleneck</td>
<td>6.9%</td>
<td>4.1%</td>
<td>Optimize network bandwidth</td>
</tr>
<tr>
<td>8</td>
<td>External Operations</td>
<td>6.9%</td>
<td>3.5%</td>
<td>Limit external operations</td>
</tr>
<tr>
<td>9</td>
<td>Database Internal Problem</td>
<td>3.4%</td>
<td>3.5%</td>
<td>Optimize database</td>
</tr>
<tr>
<td>10</td>
<td>Unknown Problem</td>
<td>2.9%</td>
<td>3.5%</td>
<td>Further diagnosis and optimization</td>
</tr>
</tbody>
</table>
Table 3: Statistics of the Weighted Average Precision, Recall, F1-score and computation time of DBSherlock and iSQUAD.

<table>
<thead>
<tr>
<th></th>
<th>Weighted Avg</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBSherlock</td>
<td>42.5</td>
<td>29.7</td>
<td>31.2</td>
<td>0.46</td>
<td>0.38</td>
</tr>
<tr>
<td>iSQUAD</td>
<td>84.1</td>
<td>79.3</td>
<td>80.4</td>
<td>0.38</td>
<td></td>
</tr>
<tr>
<td></td>
<td>↑</td>
<td>41.6</td>
<td>49.6</td>
<td>49.2</td>
<td>17.4</td>
</tr>
</tbody>
</table>

Table 4: Performance of anomaly detectors.

<table>
<thead>
<tr>
<th>Method</th>
<th>Spike F1-Score (%)</th>
<th>Running Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robust Threshold</td>
<td>98.7</td>
<td>0.19</td>
</tr>
<tr>
<td>dSPOT [41]</td>
<td>81</td>
<td>15.11</td>
</tr>
<tr>
<td>T-Test</td>
<td>92.6</td>
<td>0.23</td>
</tr>
<tr>
<td>iSST [33, 46]</td>
<td>60.7</td>
<td>6.06</td>
</tr>
</tbody>
</table>

5.2 iSQUAD Accuracy & Efficiency

We first evaluate the online root cause diagnosis & update stage. The online stage depends on its offline counterpart. We utilize iSQUAD to cluster 174 iSQs and obtain 10 clusters. How to tune the similarity score to obtain 10 clusters is discussed in §5.5. Then, using iSQUAD, we match 145 iSQs with 10 representative iSQs from the 10 clusters extracted by BCM. DBSherlock [45] is used as the comparison algorithm since it deals with database root cause analysis as well.

Table 3 lists the statistical analysis for the average accuracy of iSQUAD and DBSherlock. Weighted Average F1-score of iSQUAD is 80.4%, which is 49.2% higher than that of DBSherlock. This shows that the average precision and recall are both improved by iSQUAD significantly. Further, the computation time of iSQUAD is 0.38 second per cluster whereas that of DBSherlock is 0.46 second per cluster, improved by 17.4%. This shows that iSQUAD outperforms DBSherlock in both accuracy and efficiency. Specifically, Fig. 7 presents the precision, recall and f1-score of the two models handling the ten types of root causes. We observe that the performance of iSQUAD on different types of root causes is robust. DBSherlock, however, poorly recognizes Root Cause #2 “Host I/O Bottleneck”, #6 “Host CPU Bottleneck”, #8 “External Operations” and #10 “Unknown Problem”. This is because these types of root causes are not included in DBSherlock’s root cause types.

iSQUAD performs better than DBSherlock in four aspects. 1) DBSherlock requires user-defined or auto-generated abnormal and normal intervals of a KPI timeline. This requirement deviates from that only exact timestamps are provided here. DBSherlock’s algorithm may not produce satisfactory predicate combinations because it aims to explain KPIs in intervals, not at timestamps. Overgeneralizations from intervals are not necessarily applicable nor accurate enough to timestamps. On the contrary, iSQUAD is designed to work well with timestamps and appears to be more accurate. Moreover, DBSherlock’s way of defining and separating the intervals is problematic. It segregates two parts of an interval based on whether the difference of means of the two is over a threshold. This way is not effective when a KPI timeline fluctuates. Since such KPI fluctuations are quite common, the practicality and accuracy of DBSherlock depreciate heavily. Again, iSQUAD is robust against data turbulence because it is equipped with Anomaly Extraction which makes use of different fluctuations. 2) As explained in §5.4, DBSherlock cannot eliminate all dependencies among KPIs while iSQUAD better eradicates dependencies because of the wise choice of Confidence as the measure. 3) As we reiterate, DBSherlock fails to take DBAs’ habits into consideration. Aside of concrete predicates like CPU ≥ 40%, it overlooks that DBAs care about the anomaly types and patterns, which are exactly what we focus on. To achieve higher interpretability, unlike DBSherlock that utilizes causal models to provide plain-text explanations, iSQUAD implements the Bayesian Case Model to display understandable case-subspace representations to DBAs. To sum up, iSQUAD is interpretable with high accuracy.

5.3 Anomaly Extraction Performance

Since Anomaly Extraction is the initial and fundamental process of iSQUAD, we must guarantee that both the accuracy and efficiency are sufficiently high so that our subsequent processes can be meaningful. As previously discussed, we deploy the Robust Threshold for spike detection and T-Test for level shift detection. To evaluate the accuracy and efficiency of Anomaly Extraction, we compare the Robust Threshold with dSPOT [41] and the T-Test with iSST [33, 46], and the results are presented in Table 4. Both dSPOT and iSST are representatives of state-of-the-art spike and level-shift detectors, respectively. For our methods, we empirically set the time interval size and use grid search to pick the thresholds that generate the best F1-Scores. For the comparable methods, parameter tuning strategies are presented in their corresponding papers. Parameter analysis is left for future work.

For distinguishing spikes, the Robust Threshold gains an impressively high F1-score of around 99% whereas the result of dSPOT does not even reach 90%. Aside of that, T-Test’s accuracy, 92.6%, leads that of iSST by more than 30%. Besides, our methods are strictly proved to be more efficient. For running time, the Robust Threshold finishes one iSQ in one fifth of a second in average whereas dSPOT consumes more than 15 seconds per iSQ. Com-
paratively, T-Test spends a quarter of a second processing one iSQ while iSST needs more than 6 seconds. The main reason for this out-performance is that most state-of-the-art methods are excellent in considering a limited number of major KPIs (with seasonality) while our methods are more inclusive and general when scrutinizing KPIs. In a nutshell, the methods embedded in the Anomaly Extraction step are both accurate and efficient.

5.4 Dependency Cleansing Accuracy

We select the Confidence as the core measure to ensure that all excessive dependencies among KPIs are fully eradicated. The choice to go with the Confidence is validated in the following experiment, in which we vary parameters to choose the combination that yields the best F1-scores for all the measures. The experiment result is shown in Table 5. By comparing the precision, recall, and F1-score of the confidence and the mutual information used in DBSherlock, we find that both of them quite remarkably achieve over 90% precision. The confidence, however, also obtains extremely large percentages for the other two criteria while the mutual information significantly outperforms those of the confidence and the mutual information used in DBSherlock.

We observe that all the clustering algorithms above obtain their best accuracy scores when the resulting numbers of clusters are equal to ten (Fig. 8(b)), which is exactly the number of real-world root causes noted by DBAs. The metrics that we used are the clustering accuracy and NMI, and the results are in (c) W/o Anomaly Extraction: replace Anomaly Extraction with traditional anomaly detection in the iSQUAD framework. W/o Dependency Cleansing: skip the step of Dependency Cleansing and proceed directly from Anomaly Extraction to TOPIC. iSQUAD: the complete one we propose.

5.5 TOPIC Evaluation

### TOPIC Accuracy

We compare and contrast the performance of TOPIC and three widely-used clustering algorithms (hierarchical clustering [19], K-means [16], and DBSCAN [14]) in our scenario. For the parameters in these approaches, e.g., similarity threshold \( \sigma \) of TOPIC, the number of clusters in hierarchical clustering and K-means clustering, \( \varepsilon \) and the minimum number of points required to form a dense region (\( \text{minPts} \)) in DBSCAN, we tune them through Grid-Search in order to obtain the best accuracy [31]. We observe that all the clustering algorithms above obtain their best accuracy scores when the resulting numbers of clusters are equal to ten (Fig. 8(b)), which is exactly the number of real-world root causes noted by DBAs. The metrics that we used are the clustering accuracy and NMI, and the results are in (c) W/o Anomaly Extraction: replace Anomaly Extraction with traditional anomaly detection in the iSQUAD framework. W/o Dependency Cleansing: skip the step of Dependency Cleansing and proceed directly from Anomaly Extraction to TOPIC. iSQUAD: the complete one we propose.

### Parameter Sensitivity

In our clustering method TOPIC, the similarity \( \sigma \) is one crucial threshold that describes to what extent two KPIs’ patterns can be considered similar enough to get integrated into one. This threshold influences directly the number of clusters. We investigate the impact of this similarity threshold and the results are shown in (b) Average clustering accuracy and NMI of TOPIC under different similarity requirements. Cluster size in dotted blue line is shown in y2-axis. (c) W/o Anomaly Extraction: replace Anomaly Extraction with traditional anomaly detection in the iSQUAD framework. W/o Dependency Cleansing: skip the step of Dependency Cleansing and proceed directly from Anomaly Extraction to TOPIC. iSQUAD: the complete one we propose.

![Figure 8](image-url)

**Figure 8:** (a) Clustering ACC (accuracy) and NMI of four clustering algorithms. (b) Average clustering accuracy and NMI of TOPIC under different similarity requirements. Cluster size in dotted blue line is shown in y2-axis. (c) W/o Anomaly Extraction: replace Anomaly Extraction with traditional anomaly detection in the iSQUAD framework. W/o Dependency Cleansing: skip the step of Dependency Cleansing and proceed directly from Anomaly Extraction to TOPIC. iSQUAD: the complete one we propose.

### Table 5: Performance comparison of dependency measures.

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision (%)</th>
<th>Recall (%)</th>
<th>F1-Score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidence</td>
<td>90.91</td>
<td>100</td>
<td>95.56</td>
</tr>
<tr>
<td>MI [45]</td>
<td>100</td>
<td>40</td>
<td>57.14</td>
</tr>
<tr>
<td>Gain Ratio [20]</td>
<td>87.5</td>
<td>70</td>
<td>77.78</td>
</tr>
</tbody>
</table>

![Cluster Size](image-url)

For the parameters in these approaches, e.g., similarity threshold \( \sigma \) of TOPIC, the number of clusters in hierarchical clustering and K-means clustering, \( \varepsilon \) and the minimum number of points required to form a dense region (\( \text{minPts} \)) in DBSCAN, we tune them through Grid-Search in order to obtain the best accuracy [31]. We observe that all the clustering algorithms above obtain their best accuracy scores when the resulting numbers of clusters are equal to ten (Fig. 8(b)), which is exactly the number of real-world root causes noted by DBAs. The metrics that we used are the clustering accuracy and NMI, and the results are in (c) W/o Anomaly Extraction: replace Anomaly Extraction with traditional anomaly detection in the iSQUAD framework. W/o Dependency Cleansing: skip the step of Dependency Cleansing and proceed directly from Anomaly Extraction to TOPIC. iSQUAD: the complete one we propose.
of clusters in the y2-axis. As we gradually increase the similarity value from 0.5, both the accuracy and NMI witness a large boost initially and then remain high and stable scores when the similarity achieves 0.67. The number of clusters is ten when the similarity is in the interval from 0.65 to 0.7. This interval marks a relatively stable value of the number of clusters. Above the similarity score of 0.7, both of the accuracy and NMI begin to diverge and the number of clusters grows significantly. The two measures drop together while the accuracy plunges even more. This is because as the similarity requirement becomes overly strict, some very similar iSQs that are supposed to be together are forced to be segregated. Therefore, as the similarity overly increases, the number of member iSQs in each cluster is reduced and the clustering accuracy drops. DBAs can tune this parameter to obtain a different number of clusters in case of modifications, such as separating an existing cluster.

**Positive Effects of Previous Components on TOPIC.** We investigate the effects of the components of Anomaly Extraction and Dependency Cleansing on TOPIC whose results are shown in Fig. 8(c). Different from traditional anomaly detection that tells us only there is an anomaly or not, the Anomaly Extraction distinguishes different types of anomalies and makes use of them. From Fig. 8(c), iSQUAD with the Anomaly Extraction achieves around 90% in terms of both metrics. However, iSQUAD using traditional anomaly detection hurts the performance so much that both measure scores drop drastically by about 50%. Therefore, the Anomaly Extraction does boost iSQUAD to a very large extent. Also, iSQUAD outperforms the framework without the Dependency Cleansing by several percent for the two metrics as shown in Fig. 8(c). In summary, both Anomaly Extraction and Dependency Cleansing have positive effects on TOPIC, and the effect of the former is larger.

### 5.6 BCM Evaluation

**BCM Effectiveness.** We pay attention to both the reduction factor of BCM on KPI numbers and the overall reduced time for diagnosis. The reduction factor is calculated by comparing the numbers of BCM on KPIs before and after running iSQUAD’s offline BCM component on our datasets’ clustering results. The average reduction factor value is 35.5% which means that DBAs can take remarkably fewer KPIs for consideration. This is validated by the similarity requirement becomes overly strict, some very similar iSQs that are supposed to be together are forced to be segregated. Therefore, as the similarity overly increases, the number of member iSQs in each cluster is reduced and the clustering accuracy drops. DBAs can tune this parameter to obtain a different number of clusters in case of modifications, such as separating an existing cluster.

**Visualization Platform.** The Bayesian Case Model is embedded in the visualization platform which displays case-subspace representations of iSQ clusters along with root causes to help DBAs better understand the interconnections among iSQ clusters and their root causes. Specifically, after a DBA selects a iSQ cluster, the platform immediately shows the KPIs chosen by BCM, and it also outputs the root cause of this iSQ cluster.

### Table 6: Survey results of root cause diagnoses with and without the Bayesian Case Model.

<table>
<thead>
<tr>
<th>DBA Background</th>
<th># of DBAs</th>
<th># of Correct Answers (%) w/ BCM</th>
<th># of Correct Answers (%) w/o BCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beginner</td>
<td>14</td>
<td>51.4</td>
<td>34.3</td>
</tr>
<tr>
<td>Intermediate</td>
<td>14</td>
<td>65.7</td>
<td>48.8</td>
</tr>
<tr>
<td>Advanced</td>
<td>18</td>
<td>84.4</td>
<td>62.2</td>
</tr>
</tbody>
</table>

**User Study.** We conduct a formal user study survey to quantitatively evaluate BCM. We randomly distribute surveys to DBAs with various levels of database background (the beginner, intermediate, and advanced). The survey contains a dozen of four-choice questions that present either KPIs selected with BCM or without BCM (i.e., selected arbitrarily) and ask for corresponding root causes. We calculate the percentage of correct responses w.r.t each group of DBAs and observe that the accuracy with BCM surpasses that without BCM by 18.7% in average for all DBAs as shown in Table 6. In particular, this performance improvement is more significantly shown by DBAs who have advanced database knowledge.

### 6. CASE STUDY AND DISCUSSION

**Case Study.** To study how iSQUAD speeds up the root cause diagnosis of iSQs, we randomly pick a small fraction of iSQs, of which iSQUAD does not directly deliver diagnostic results to DBAs. In the following case, we compare iSQUAD’s result with one experienced DBA’s manual diagnosis.

**Fig. 9** shows the timeline of a database failure. At 11:50, the KPI of mysql.qps drastically dropped and a large number of active sessions started to accumulate. After approximately nine minutes at 11:59, the service was completely down. An experienced DBA turned to inspect this failure soon after the alert was flagged. Having spent almost fifteen minutes of manual diagnosis, the DBA decided that this was a database internal problem (may be a hang). At 12:17, the DB instance was recovered by the DBA.

At 11:50, a burst of iSQs emerged and caused iSQUAD to initiate the analysis. Forty seconds later on the side of our framework, iSQUAD quickly recognized that mysql.qps appeared to be a level shift-down and mysql.active-session a spike. (Please note that the KPIs are not limited to these two and other KPIs also demonstrated diverse patterns.) Then, based on these symptoms (to name a few as example) to match with clusters, iSQUAD proposed that the root cause in this case was “database internal problem” based on analysing KPI behaviors. To summarize, the whole process of manual diagnosis took eighteen minutes in total without iSQUAD, while iSQUAD was proved much faster. Therefore, iSQUAD can not only save the time of root cause analysis, but also accurately pinpoint the correct root causes.

**Multiple Root Causes.** We may discuss both dependent root causes and independent ones separately. For dependent root causes, e.g., “instance CPU intensive workload” and “instance I/O intensive workload” in Table 2, DBAs label them as one type of root causes, because CPU and I/O resources are often closely related and we do not bother to separate them. If two root causes always occur together, more in-depth causes shall be explored. For independent root causes, according to DBAs’ experience, the chance of multiple root causes occurring simultaneously is very small since it is a joint probability of multiple events with small probabilities.

**Generality of iSQUAD.** We discuss the generality of iSQUAD in two aspects. First, we evaluate iSQs using various business services of Alibaba Group, e.g., online retail, enterprise collaboration and logistics, which guarantee the diversity of studied services. Second, the input data used in iSQUAD, e.g., iSQs and KPIs, are
general and common so the framework of iSQUAD is applicable to root cause analysis of iSQs for diverse types of databases.

**Root Causes to Actions.** To better facilitate the practical usage of iSQUAD, we recommend problem-solving procedures of tackling three main categories of iSQ root causes as shown in Table 2.(1) Scaling (#1, #2, #3, #5, #6): For those problems of instances or physical machines, we suggest that the resources of the anomalous instances or physical machines can be scaled up or scaled out automatically. Besides, the root causes of anomaly workloads are further classified into different categories, i.e., CPU, I/O, memory, and network, based on which we can give more specific suggestions. (2) Limiting (#4, #8): For the problems caused by accompanying slow queries (that alter tables with considerable rows) or external operations, such as dumping table or database backup, we can limit their resources. For example, for insertions, deletions, or updates, we recommend that DBAs apply rate-limiting thresholds onto these slow queries. (3) Optimizing (#7, #9, #10): If a root cause belongs to database internals or unknown problems, we suggest DBAs optimize the database accordingly. For example, a case shows two tables joining with hundreds of billions of rows in the temporary directory, which causes disk to be full and a burst of iSQs. In this case, we suggest modifying the database kernel to limit temporary directory resources. We believe that these actions are adequate for most common scenarios. As a future work, we aim to develop, on top of iSQUAD, a more versatile and capable framework that automates fault fixes and system recoveries.

7. RELATED WORK

**Slow Query Analysis.** Slow query analysis and optimization have been extensively studied. General approaches involve data-driven automatic analyses and optimizations of databases and queries. For databases, several previous studies [13, 28, 40] aim to automate indexing modifications to achieve better performance, and one study addresses the issue of tuning database parameters with machine learning algorithms [32].

**Anomaly Extraction.** Past anomaly detection algorithms generally output binary results i.e., either “normal” or “anomalous”. In the literature, there exist various anomaly detectors, such as Oppenprentice [31], DSPOT [41] and iSST [33, 46]. Also, some corpora develop anomaly detectors, e.g., Yahoo’s EGADS [27], Twitter’s S-H-ESD [21], and Netflix’s RPCA [17]. Different from them, our Anomaly Extraction returns KPI states, i.e., normal or one of the discussed anomaly categories, rather than limited binary results.

**Clustering Algorithm.** Some query-related clustering algorithms provide insights. K-Shape clustering [38], built on [3], clusters queries based on KPI timelines’ shapes. This method is off from our scenario since we focus on one timestamp across all KPIs while K-Shape allows a time lag between two similar shapes. Such a latency may render two irrelevant queries together and incur accuracy loss. Next, the workload compression technique in [32] is similar to our work. It computes the similarity of workload features based on the cosine similarity. One drawback is that it loses the information of KPI types, which are crucial for determining query behaviors. By contrast, our TOPIC considers both KPI types and anomaly patterns to cluster queries in a rigorous way. Further, TOPIC does not modify the cluster centers, i.e., anomaly patterns, of existing clusters like [32], because patterns, which are integrated when merged, are stable unlike templates in [32] that vary with time, so the clusters converge more quickly.

**Root Cause Diagnosis.** PerfXplain [22] helps explain abnormal behaviors of MapReduce jobs but does not fit our scenario, because iSQUAD is designed for iSQ analyses while PerfXplain cannot deal with iSQ. Our method utilizes clustering to help identify case-related root causes rather than directly giving despite clauses that require relevant identified task pairs. The predicate-based explanations of PerfXplain are similar to those of DBSherlock [45], which are less accurate than our method’s output. DBSherlock concentrates on the exact values of KPIs, but ignores real actions of DBAs who also care about categories of anomalies. A concrete KPI figure can imply only whether an indicator is anomalous, whereas our Anomaly Extraction method can well inform DBAs of the definite category of the anomaly, which is much more useful for real-world root cause diagnoses as demonstrated in our experiments. Moreover, DBSherlock resembles a more general OLTP tool while iSQUAD is for iSQ root cause diagnoses only. Besides, iSQUAD is trained with real-life datasets as opposed to DBSherlock’s generated datasets. Furthermore, probabilistic graphical models are implemented in [18] for causal inference to analyze root causes, but they require excessive user operation per execution, which is not even feasible in our scenario considering our dataset size. A concept of “fingerprints” [6] is introduced to help detect datacenter crisis, by checking if KPIs are over a threshold and by comparing distances between online fingerprints and existing ones. This anomaly detector and similarity comparison standard are both too simplistic compared to Anomaly Extraction and the CalculateSimilarity function of TOPIC in iSQUAD. Moreover, it is applicable to only huge datacenters, whereas ours is to diagnose iSQs running in database instances and physical machines.

8. CONCLUSION

In this work, we identify the problem of intermittent slow queries (iSQs) in real-world cloud databases. A large number of detrimental iSQs are generated in cloud databases, but DBAs cannot diagnose them one by one, since this is very labor-intensive and time-consuming. To deal with this dilemma, we present iSQUAD, a framework for iSQ root cause diagnoses, which contains several key components, i.e., Anomaly Extraction, Dependency Cleansing, Type-Oriented Pattern Integration Clustering, and Bayesian Case Model. To a very large extent, iSQUAD can help DBAs with online root cause diagnoses by accurately and efficiently analyzing, processing, classifying online iSQs and outputting highly precise root cause diagnostic results. Extensively tested in experiments on Alibaba’s real-world datasets, iSQUAD is strictly proved to across-the-board outperform the state-of-the-art root cause diagnosers to the best of our knowledge. A prototype of iSQUAD is now deployed in Alibaba OLTP Database to surveil and handle iSQs.

9. ACKNOWLEDGEMENTS

We appreciate Jun (Jim) Xu from Georgia Tech, Junjie Chen from Tianjin University and Kaixin Sui for their valuable feedback. We thank our shepherd and the anonymous reviewers for their thorough comments and helpful suggestions. This work has been supported by Alibaba Group through Alibaba Innovative Research Program. Dan Pei and Minghua Ma have also been partially supported by the National Key R&D Program of China under Grant No. 2019YFB1802504 and the National Research Center for Information Science and Technology (BNRist) key projects. Shenglin Zhang has also been partially supported by the National Natural Science Foundation of China under Grant No. 61902200 and the China Postdoctoral Science Foundation under Grant No. 2019M651015.
10. REFERENCES


