Diagnosing Root Causes of Intermittent Slow Queries in Cloud Databases

Minghua Ma[*1,2], Zheng Yin[2], Shenglin Zhang[†3], Sheng Wang[2], Christopher Zheng[1], Xinhao Jiang[1], Hanwen Hu[1], Cheng Luo[1], Yilin Li[1], Nengjun Qiu[2], Feifei Li[2], Changcheng Chen[2] and Dan Pei[1]

[1]Tsinghua University, {mmh16, luo-c18, liyilin16}@mails.tsinghua.edu.cn, peidan@tsinghua.edu.cn
[2]Alibaba Group, {yinzheng.yz, sh.wang, nengjun.qnj, lifeifei}@alibaba-inc.com
[3]Nankai University, zhangsl@nankai.edu.cn

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 Diagnoser, 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:
M. Ma, Z. Yin, S. Zhang, S. Wang, C. Zheng, X Jiang, H. Hu, C. Luo, Y. Li, N. Qiu, F. Li, C. Chen and D. Pei. Diagnosing Root Causes of Intermittent Slow Queries in Cloud Databases. *PVLDB*, 13(8): 1176-1189, 2020.
DOI: https://doi.org/10.14778/3389133.3389136

[*]Work was done while the author was interning at Alibaba Group.
[†]Work was done while the author was a visiting scholar at Alibaba Group.

Proceedings of the VLDB Endowment, Vol. 13, No. 8
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3389133.3389136

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, 34, 42]. 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 led 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, 6, 9, 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 one 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 [36].

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 [26].

To address the aforementioned challenges, we design **iSQUAD** (**I**ntermittent **S**low **QU**ery **A**nomaly **D**iagnoser), 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 overcomes 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. §4 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 and our future work. §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 [9] 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 [7, 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 X_t) is an iSQ if and only if $X_t > z$ and $P(X_i > z) < \epsilon$, where $1 \leq t, i \leq T$ (T is the total

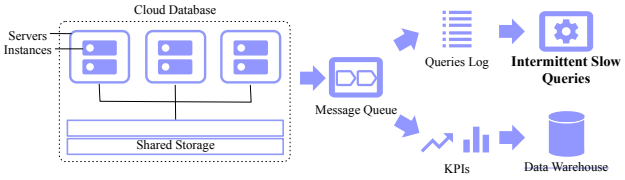
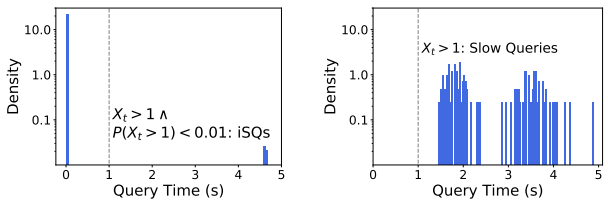


Figure 1: The architecture of the data collection system for Alibaba OLTP Database.



(a) PDF of iSQs (b) PDF of slow queries

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

number of Q's recent occurrences), z is slow query threshold, and ϵ is iSQ probability threshold. For interactive transactional workloads on Alibaba Database, DBAs empirically set $z = 1s$, $\epsilon = 0.01$, and $T = 10^4$. 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 ϵ. 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

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

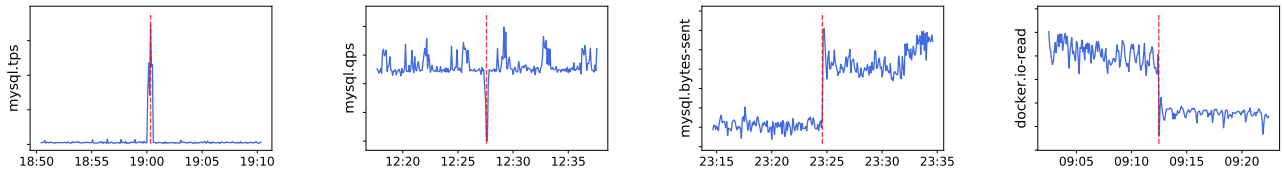
	Type	# KPIs	Example
Instance (45)	CPU	2	*docker.cpu-usage*
	I/O	15	*mysql.io-bytes*
	Workload	13	*mysql.tps*
	TCP RT [9]	12	*tcp.rt99*
	Memory	3	*mysql.buffer-pool-reads*
Physical Machine (14)	CPU	6	*cpu.usage*
	I/O	4	*io.wait-usage*
	Network	4	*net.receive-usage*

(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 I/O-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-



(a) Spike Up (b) Spike Down (c) Level Shift Up (d) Level Shift Down

Figure 3: Four types of anomalies. A red dash line signals an occurrence of an iSQ. (The exact values of KPIs are hidden for confidential reasons.)

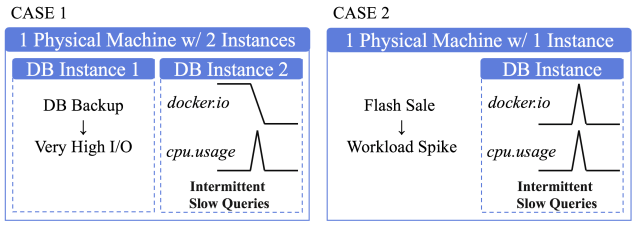


Figure 4: An example – two typical cases of intermittent slow queries (iSQs).

tual. For example, anomalies on instances (*docker.cpu-usage*) are highly possible to incur their anomalous counterparts on physical machines (*cpu.usage*), whereas problematic KPIs on physical machines (*cpu.usage*) may not always see corresponding problems on their instances' KPIs (*docker.cpu-usage*).

Similar KPI patterns are correlated with the same root cause. DBAs summarize ten types of root causes in cloud database based on performance issue records (Table 2). In each type of root causes, KPI symptoms of failure records are similar. KPIs in the same type can substitute each other, but their anomaly types are constant. For example, "mysql.update-ps" and "mysql.delete-ps" are in the same group of "mysql workload per second (ps)". They both indicate the same root cause – workload anomaly. As a result, when performing RCA, DBAs do not have to care about whether the anomaly is caused by the SQL of "update" or "delete".

A cloud database possesses features, such as instance migrations, capacity expansions, host resource sharing, or storage decoupling, that can cause iSQs. We explain how root causes are related to the features of a cloud database: in a cloud database service, a physical machine can host several database instances, which may lead to resource contentions, such as host CPU, I/O, network bottleneck. Besides, intensive workload can occur more frequently. For example, intensive workload is the root causes of many iSQs. Because of storage decoupling, the low latency of data transmissions cannot always be guaranteed between computation and storage nodes [29]. Therefore, queries with large I/O demand may cause I/O bottlenecks and accompanying slow SQLs.

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.

Limitation of existing solutions: Previous works [45] reproduce root causes in testbed experiments rather than label root causes. In our case, however, simply reproducing known root causes in a testbed experiment is not feasible because it is hard to mimic such a large number of machines, instances, activities, interactions, *etc.* Besides, datasets of custom workloads are usually not in good conditions as for availability and maintenance. Aside of the complexity of making a facsimile of the original scenario for the experiment, even if we manage to reproduce the past scenarios, experiment statistics are expected to be prohibitive to process.

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-

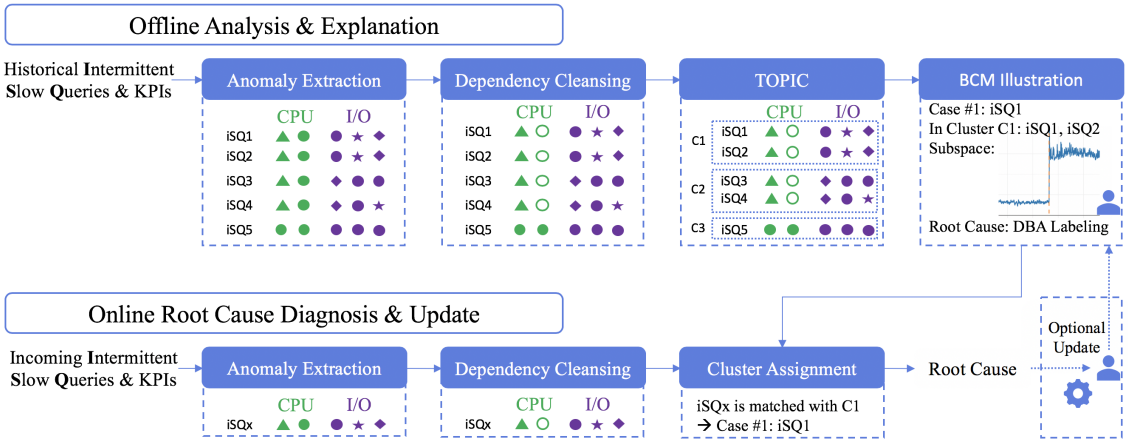


Figure 5: Framework of iSQUAD.

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 [26]. 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 (**I**ntermittent **S**low **QU**ery **A**nomaly **D**iagnoser), 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 **T**ype-**O**riented **P**attern **I**ntegration **C**lustering (**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.

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

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.

KPI Type	CPU	I/O	Network	Workload	...
iSQ1	▲ ●	● ★ ◆ ▲ ▲ ● ★	● ★ ★ ●	▼ ● ▼ ●	...
iSQ2	▲ ●	● ◆ ★ ▲ ▲ ◆ ★	◆ ★ ★ ●	▼ ● ▼ ●	...
Similarity	100%	57% (4/7)	75% (3/4)	100%	...

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.

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** encapsulates 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:

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

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:

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

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"

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 σ) 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 "X 0 0 0 0" & "0 0 0 0 0", where X denotes an arbitrary anomaly that can be of any type. The former pattern denotes when one KPI is somehow anomalous while the later one is fully safe and sound, and apparently they are distinct patterns in our scenario. These two patterns, however, tend to be clustered into the same group if we do not eliminate the all-zero pattern from `D` before the iteration. "All-zero-pattern" means there are no anomalies in KPIs. These issues are almost infeasible to diagnose due to lack of anomaly symptoms from KPIs. We focus on root causes that can be explained or reflected by KPIs, and leave all-zero-pattern issues as future work.

To cluster iSQs based on patterns, we first store `D`'s patterns into a KD-tree [4], a very common approach to searching for the nearest element in clustering (Line 9). For each pattern i in `D`, the function finds its nearest pattern j (Line 11). If both i and j are still inside `D` and their patterns are similar (how to choose a reasonable similarity threshold is introduced in §5.5), the function merges two patterns into a new one (Line 10 to 14). Specifically, when we merge two anomaly patterns, we first check their numbers of corresponding iSQs in the dictionary `D`. The pattern with the larger number is reserved, while the one with the smaller number is dropped with its corresponding iSQs added to the former pattern's counterpart. As the precondition for this merging is the similarity checking, the two iSQs are already very similar. Therefore, the merging policy in fact has quite limited impact on the final result, and this speculation is confirmed by our observation. The iteration terminates when the size of `D` no longer changes (Line 15 to 16).

Note that, to improve computational efficiency, we use a dictionary `D` to gather all the patterns first and then combine identical ones. Also, for each pattern, we use a KD-tree to select the pattern that satisfies the similarity check with the highest similarity score and continue adjusting, so that the results can be more accurate than its greedy counterpart. The time complexity is bounded by $O(n \log n)$, where n is the number of different patterns inside the dictionary `D` and is always no larger than the number of iSQs. Therefore, this algorithm's running time is positively associated with the number of initial patterns.

Algorithm 1: TOPIC

```
   Data: Intermittent slow queries under clustering
         S ← [iSQ_index : pattern]
   Input: Similarity threshold σ
   Output: Clusters' dictionary C
1  C, D ← empty dictionary
   /* Reverse S into D: the indices and values
      of D are respectively the values and
      clustered indices of S                    */
2  for iSQ_index in S do
3      add iSQ_index to D[S[iSQ_index]]
4  if all-zero pattern exists in D then
5      C ← D.pop(all-zero pattern)
6  C ← C + PatternCluster(D)
7
8  PatternCluster (D):
9      KDTree(D.patterns)
10     for i in D.patterns do
           /* find the nearest pattern to i     */
11         j ← KDTree.query(i)
           /* i or j may be merged (Line 14)    */
12         if i & j in D and CalculateSimilarity(i, j) > σ then
               /* k is either i or j whichever has
                  a larger number of corresponding
                  iSQs                          */
13             k ← arg max_{l∈{i,j}} D[l].length
14             D[k] ← D.pop(i) + D.pop(j)
       /* recursively cluster unmerged patterns
          */
15     if D remains unchanged then
16         return D
17     return PatternCluster(D)
18
19 CalculateSimilarity (Pattern x, Pattern y):
20     s ← 0, κ ← the set of all KPI categories
21     for t in κ do
22         α is a segment of x w.r.t. t
23         β is a segment of y w.r.t. t
24         s += SimpleMatchingCoefficient(α, β)^2
25     return sqrt(s / κ.length)
```

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. 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 straight-forward 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 §4.1.1. 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 more able to understand the result of clusters, and we can thus deliver more suggestive information to DBAs.

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.

No.	Root Cause	Offline	Online	Solution
1	Instance CPU Intensive Workload	27.6%	34.5%	Scale up instance CPU
2	Host I/O Bottleneck	17.2%	17.2%	Scale out host I/O
3	Instance I/O Intensive Workload	0.9%	15.8%	Scale up instance I/O
4	Accompanying Slow SQL	8.6%	9.0%	Limit slow queries
5	Instance CPU & I/O Intensive Workload	8.1%	4.8%	Scale up instance CPU and I/O
6	Host CPU Bottleneck	7.5%	4.1%	Scale out host CPU
7	Host Network Bottleneck	6.9%	4.1%	Optimize network bandwidth
8	External Operations	6.9%	3.5%	Limit external operations
9	Database Internal Problem	3.4%	3.5%	Optimize database
10	Unknown Problem	2.9%	3.5%	Further diagnosis and optimization

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 σ) 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.

5. EVALUATION

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.

5.1 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 a certain number of instances running on the physical machines of Alibaba Database, which simultaneously host hundreds of thousands of instances. Next, for each instance, we select one iSQ at each timestamp. That is, we choose only one iSQ for each unique instance-host-timestamp signature. This selection guarantees that we can safely assume that choosing one for analysis is sufficient to represent its group of homogeneous queries. In sum, we obtain three datasets of iSQs, one for an arbitrary day and two for one week each. These datasets are random. In this way, our analyzed iSQs represent almost all the types and typical behaviors of iSQs given the variety of the dataset size and time span.

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 analyze the aforementioned 59 KPIs and discover ten underlying dependencies among them. For both TOPIC and BCM, DBAs carefully label iSQs' ten root causes.

Considering the large number of iSQs and labeling overhead in §2.3, DBAs select one day's unique iSQs (319 in total) from the datasets for labeling purpose. Recall that iSQUAD comprises unsupervised learning with both *offline analysis & explanation* and *online root Cause diagnosis & update*. We divide labeled iSQs and use the first 55% for offline clustering, and the other 45% for online testing. This division is similar to the setting of training and test in supervised learning. For the iSQUAD framework, each one of the root causes is shown in Table 2. This ground truth set is considerable in size because of labeling overhead and is comparable with what was used in DBSherlock [45]. The details of manual labeling are as follows: experienced DBAs spend one week analyzing 319 of iSQs one by one randomly chosen from three datasets. They label the root causes of iSQs as ten types as shown in Table 2. We adopt these label as the ground truth for offline clustering part §5.5 and online diagnosis §5.2. This setting is comparable with that of DBSherlock, and is unbiased towards iSQUAD.

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.

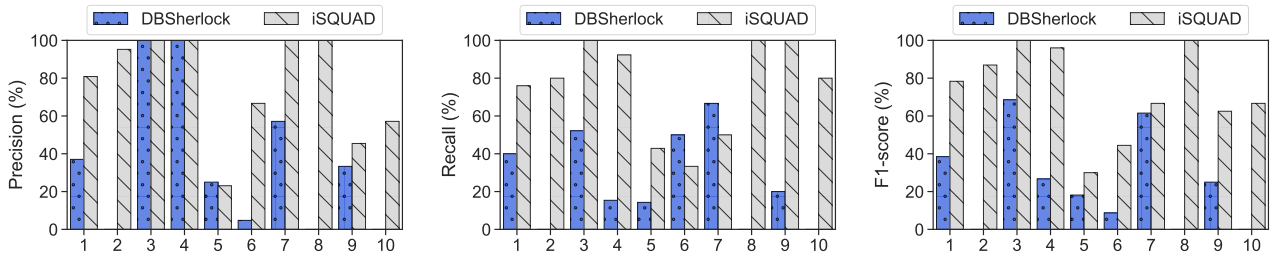
F1-Score: the harmonic mean of precision and recall. It is used to evaluate the performance of iSQUAD online diagnoses (§5.2), Anomaly Extraction (§5.3) and Dependency Cleansing (§5.4).

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. (§5.5)

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. (§5.5)

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.



(a) Precision (b) Recall (c) F1-score

Figure 7: Performance of DBSherlock [45] and iSQUAD in online diagnoses of ten types of root causes.

Table 3: Statistics of the Weighted Average Precision, Recall, F1-score and computation time of DBSherlock and iSQUAD.

Weighted Avg.	Precision	Recall	F1-score	Time
DBSherlock	42.5	29.7	31.2	0.46
iSQUAD	84.1	79.3	80.4	0.38
⇑	41.6	49.6	49.2	17.4

Table 4: Performance of anomaly detectors.

	Method	F1-Score (%)	Running Time (s)
Spike	Robust Threshold	98.7	0.19
	dSPOT [41]	81	15.11
Level Shift	T-Test	92.6	0.23
	iSST [33, 46]	60.7	6.06

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.

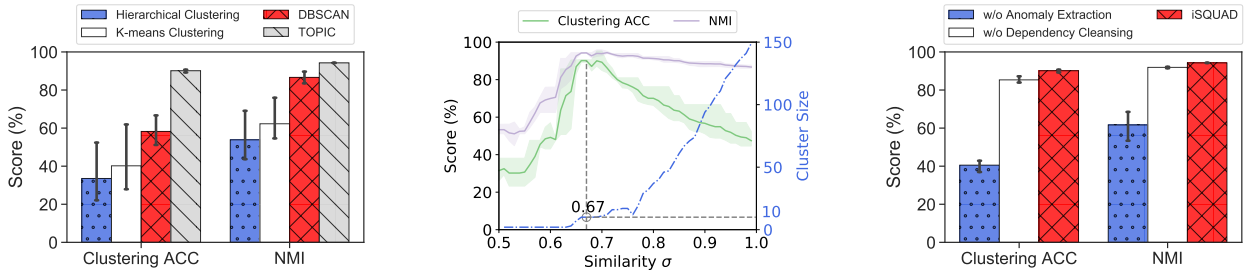
Table 3 lists the statistical analysis for the average accuracy of iSQUAD and DBSherlock. Weighted Average F1-score of iSQUAD is 80.4%, which is 49.2% higher than that of DBSherlock. This shows that the average precision and recall are both improved by iSQUAD significantly. Further, the computation time of iSQUAD is 0.38 second per cluster while 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. Over-generalizations 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 $CPU \geq 40\%$, 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-



(a) TOPIC Accuracy (b) Parameter Sensitivity (c) Previous Components

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.

Method	Precision (%)	Recall (%)	F1-Score (%)
Confidence	90.91	100	95.24
MI [45]	100	40	57.14
Gain Ratio [20]	87.5	70	77.78

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 performs poorly. The recall of the mutual information is only 40% because it fails to spot or capture a large proportion of the underlying dependencies, and the F1-score is dragged down as a consequence. By comparing the scores of the confidence and the gain ratio, we find that the confidence also significantly outperforms the latter in terms of all the three scores. Therefore, it is indeed an appropriate decision to detect and eliminate KPI dependencies with the Confidence measure.

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 σ of TOPIC, the number of clusters in hierarchical clustering and K-means clustering, ε and the minimum number of points required to form a dense region (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 Fig. 8(a). (The shown results are the best scores generated by tuning parameters.) For the first metric, both the hierarchical and K-means clustering obtain less than half of the score of TOPIC. Also, TOPIC's accuracy leads that of DBSCAN by more than 30%. For the second metric, TOPIC's NMI outperforms those of hierarchical and K-means clustering by around 35%, and outperforms that of DBSCAN by about 5%. In sum, TOPIC is promising to cluster iSQs.

TOPIC is preferable because it considers both KPI types and anomaly patterns. This mechanism is intuitive for clustering iSQs with multiple KPIs. Here we analyze the reasons why the three traditional clustering algorithms (hierarchical clustering, K-means, and DBSCAN) are not as good as TOPIC in our settings. 1) Hierarchical clustering is very prone to outlier effect. When it encounters a new iSQ's pattern, the hierarchical clustering may categorize it into an outlier if it is very different from existing ones instead of constructing a new cluster for it like what TOPIC does. Besides, the number of clusters needs to be preset. 2) K-means clustering requires a pre-defined number of clusters as well. Plus, it is highly dependent on initial iSQ patterns so it is unstable. 3) TOPIC is to some extent similar to DBSCAN in the sense that they do not require preset numbers of clusters and they cluster elements w.r.t. key thresholds. Nonetheless, after scrutinizing the shapes of clusters produced by DBSCAN, we notice that its resulting clusters are heavily skewed and scattered, *i.e.*, some cluster has far more iSQs (most of which are of all-zero patterns) than its peers do and many of other clusters may house only one or two iSQs. This result is meaningless and cannot be used for further analysis. On the contrary, the clusters generated by TOPIC are to some extent reasonably distributed and are much better-shaped. Clusters like those are more able to convey information of groups of iSQs.

Parameter Sensitivity. In our clustering method TOPIC, the similarity σ 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 Fig. 8(b). In this figure, we use three datasets' Clustering Accuracy and NMI with error bar to analyze the robustness of the threshold. Besides, we also plot the numbers

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

DBA Background	# of DBAs	# of Correct Answers (%) w/ BCM	w/o BCM
Beginner	14	51.4	34.3
Intermediate	14	65.7	48.8
Advanced	18	84.4	62.2

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 retain 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 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 significant reduced diagnosis time. Given the number of iSQs in our datasets, DBAs spend about 80 min diagnosing ten cases produced by iSQUAD. On the contrary, they need a whole week's working time (approximately 2,400 min) to label 319 iSQs (*i.e.*, 7.5 min per iSQ in average) in the traditional way, *i.e.*, case-by-case diagnosis without iSQUAD. Therefore, diagnosing root causes of iSQs using iSQUAD can be fifteen times faster than the traditional diagnosis.

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.

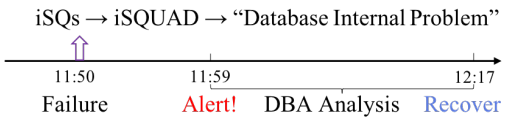


Figure 9: Time line of a database failure case.

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 retails, 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 [42]. For query optimization, boosting queries by deep learning is introduced in [25, 34]. Neither of the above touches the field of iSQs. Our work is the first to reduce negative effects of iSQs on database systems.

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 Opprentice [31], dSPOT [41] and iSST [33, 46]. Also, some corporations 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 [5], 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

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD*, volume 22, pages 207–216.

[2] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM*, pages 440–453.

[3] C. Bell, M. Kindahl, and L. Thalmann. *MySQL high availability: tools for building robust data centers.* " O'Reilly Media, Inc.", 2010.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Proceedings of the Communications of the ACM*, 18(9):509–517, 1975.

[5] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 1994 ACM SIGKDD*, volume 10, pages 359–370, 1994.

[6] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen. Fingerprinting the datacenter: automated classification of performance crises. In *Proceedings of the 2010 EUROSYS*, pages 111–124.

[7] D. Burleson. Find slow oracle sql. `http://www.dba-oracle.com/t_find_slow_oracle_sql.htm`, 2015.

[8] D. Cai, X. He, X. Wang, H. Bao, and J. Han. Locality preserving nonnegative matrix factorization. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[9] W. Cao, Y. Gao, B. Lin, X. Feng, Y. Xie, X. Lou, and P. Wang. Tcprt: Instrument and diagnostic analysis system for service quality of cloud databases at massive scale in real-time. In *Proceedings of the 2018 ACM SIGMOD*, pages 615–627.

[10] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *Proceedings of the 2008 OSDI*, volume 8, pages 117–130.

[11] N. Chinchor. Muc-4 evaluation metrics. In *Proceedings of the 4th conference on Message understanding*, pages 22–29. Association for Computational Linguistics, 1992.

[12] L. Columbus. 83% of enterprise workloads will be in the cloud by 2020. `https://www.forbes.com/sites/louiscolumbus/2018/01/07/83-of-enterprise-workloads-will-be-in-the-cloud-by-2020#70765a696261`, 2019.

[13] S. Das, M. Grbic, and e. Ilic. Automatically indexing millions of databases in microsoft azure sql database. In *Proceedings of the 2019 ACM SIGMOD*.

[14] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 1996 ACM SIGKDD*, volume 96, pages 226–231.

[15] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proceedings of the 1999 ICML*, volume 99, pages 124–133.

[16] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[17] W. J. Netflix surus github, online code repos. `https://github.com/Netflix/Surus`, 2015.

[18] V. Jeyakumar, O. Madani, A. Parandeh, A. Kulshreshtha, W. Zeng, and N. Yadav. Explainit!–a declarative root-cause analysis engine for time series data. In *Proceedings of the 2019 SIGMOD*, pages 333–348.

[19] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[20] A. G. Karegowda, A. Manjunath, and M. Jayaram. Comparative study of attribute selection using gain ratio and correlation based feature selection. volume 2, pages 271–277.

[21] A. Kejariwal. Twitter engineering: Introducing practical and robust anomaly detection in a time series. `https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.html`, 2015.

[22] N. Khoussainova, M. Balazinska, and D. Suciu. Perfxplain: debugging mapreduce job performance. *PVLDB*, 5(7):598–609, 2012.

[23] B. Kim, C. Rudin, and J. A. Shah. The bayesian case model: A generative approach for case-based reasoning and prototype classification. In *Proceedings of the 2014 NeurIPS*, pages 1952–1960.

[24] M. Kim, R. Sumbaly, and S. Shah. Root cause detection in a service-oriented architecture. *ACM SIGMETRICS Performance Evaluation Review*, 41(1):93–104, 2013.

[25] T. Kraska, M. Alizadeh, A. Beutel, E. H. Chi, J. Ding, A. Kristo, G. Leclerc, S. Madden, H. Mao, and V. Nathan. Sagedb: A learned database system.

[26] M. Kuhn and K. Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.

[27] N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 2015 ACM SIGKDD*, pages 1939–1947.

[28] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *PVLDB*, 9(3):204–215, 2015.

[29] F. Li. Cloud-native database systems at alibaba: Opportunities and challenges. 12(12):2263–2272, 2019.

[30] G. Linden. Akamai online retail performance report: Milliseconds are critical. `http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html`, 2006.

[31] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proceedings of the 2015 IMC*, pages 211–224.

[32] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 SIGMOD*, pages 631–645.

[33] M. Ma, S. Zhang, D. Pei, X. Huang, and H. Dai. Robust and rapid adaption for concept drift in software system anomaly detection. In *Proceedings of the 2018 IEEE ISSRE*, pages 13–24.

[34] R. Marcus and O. Papaemmanouil. Towards a hands-free query optimizer through deep learning. *arXiv preprint arXiv:1809.10212*, 2018.

[35] J. C. Mogul and J. Wilkes. Nines are not enough: Meaningful metrics for clouds. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 136–141. ACM, 2019.

[36] B. Mozafari, C. Curino, A. Jindal, and S. Madden. Performance and resource modeling in highly-concurrent

oltp workloads. In *Proceedings of the 2013 ACM SIGMOD*, pages 301–312.

[37] MySQL. Mysql slow query log. https://dev.mysql.com/doc/refman/8.0/en/slow-query-log.html, 2019.

[38] J. Paparrizos and L. Gravano. k-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD*, pages 1855–1870.

[39] A. Pettitt. A non-parametric approach to the change-point problem. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 28(2):126–135, 1979.

[40] K. Schnaitter and N. Polyzotis. Semi-automatic index tuning: Keeping dbas in the loop. *PVLDB*, 5(5):478–489, 2012.

[41] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet. Anomaly detection in streams with extreme value theory. In *Proceedings of the 2017 ACM SIGKDD*, pages 1067–1075.

[42] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM SIGMOD*, pages 1009–1024.

[43] M. Watson. Performance tuning in sql server tutorial: Top 5 ways to find slow queries. https://stackify.com/performance-tuning-in-sql-server-find-slow-queries, 2017.

[44] Wikipedia. Simple matching coefficient. https://en.wikipedia.org/wiki/Simple_matching_coefficient, 2018.

[45] D. Y. Yoon, N. Niu, and B. Mozafari. Dbsherlock: A performance diagnostic tool for transactional databases. In *Proceedings of the 2016 ACM SIGMOD*, pages 1599–1614.

[46] S. Zhang, Y. Liu, D. Pei, and et.al. Rapid and robust impact assessment of software changes in large internet-based services. In *Proceedings of the 2015 ACM CONEXT*, page 2.

[47] N. Zhao, L. Zhang, B. Du, Q. Zhang, J. You, and D. Tao. Robust dual clustering with adaptive manifold regularization. *Proceedings of the 2017 IEEE TKDE*, 29(11):2498–2509.