Diagnosing Root Causes of Intermittent Slow Queries in Large-Scale Cloud Databases

Minghua Ma[1], 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*
[2]*Alibaba Group*
[3]*Nankai University*

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**, **I**ntermittent **S**low **QU**ery **A**nomaly **D**iagnoser, the first 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 very 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, which work sequentially. 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 offline stage, unless a new type of iSQs emerges at online stage. Our evaluations on real-world datasets from Alibaba OLTP Database show that iSQUAD achieves an iSQ root cause diagnosis accuracy of 88%, and outperforms all existing diagnostic tools in terms of both 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 Large-Scale Cloud Databases. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

[*]Shenglin Zhang is the corresponding author. Work was performed while the author was a visiting scholar at Alibaba Group. Email: *zhangsl@nankai.edu.cn*
[†]Work was performed while the author was interning at Tsinghua University.

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Database management systems (DBMSes), as well as the growing cloud database services (such as AWS RDS, Azure Cloud 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, PostgreSQL, automatically log detailed information of those queries whose completion time is over a user-defined threshold [8, 35, 40], *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, 30, 32, 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**).

Here we illustrate two typical cases where iSQs can occur. Consider the first case shown in Fig. 1, in which two database 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 which is unavoidably resource-demanding, and it consequently triggers an anomaly associated with I/O (reflected in one or more I/O-related Key Performance Indicators, or KPIs for short). 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 running 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.

In practice, the iSQs might be more hazardous than other slow queries for database users. Since iSQs' facsimiles or similar queries at other times are not slow, service developers and customers expect them to be responsive as others, where sudden increases of latency have huge impact. 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 [28]. In contrast, other types of slow queries (*e.g.*, complicated analytics) are usually non-interactive and tolerable.

Moreover, this iSQ issue is more detrimental under the circumstance of large-scale cloud databases. First, iSQ occurrences become more common. Multiple database instances may reside on the same physical machines for better utilization, which in turn causes inter-database resource contentions. Second, root causes of iSQs greatly vary in cloud databases. Infrastructures of cloud databases are more complex than those of on-premise databases, making it harder 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. It is prohibitive to diagnose the root cause of each iSQ with heavy human investigations and interventions. Moreover, roughly 83% of enterprise workloads are forecast to be in the cloud by 2020 [12]. This tend makes it critical to efficiently diagnose the root causes of iSQs.

In this work, we aim to diagnose root causes of iSQs in large-scale cloud databases with minimal human intervention. In fact, this is a Root Cause Analysis (RCA) problem that is a broad and general concept [2, 6, 10, 18]. Traditional RCA, however, does not have the capability of specifically diagnosing the root causes of iSQs. For instance, using system monitoring data, *i.e.*, single KPI alone (or a single type of KPIs), usually cannot pinpoint iSQs' root causes [11]. It is also infeasible to enumerate and verify all possible causalities between anomalous KPIs and root causes individually [6, 34, 44]. Furthermore, even if two iSQs have the identical set of anomalous KPIs (but with distinct anomaly behaviors), their root causes can differ.

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, shift-up, shift-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** (**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.

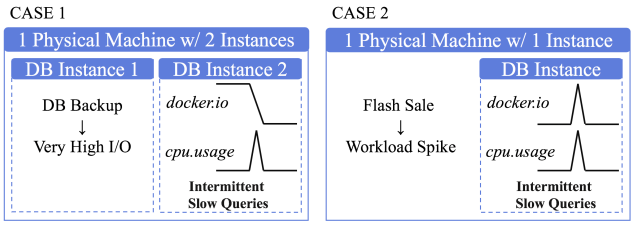


Figure 1: An example – two typical cases of intermittent slow queries (iSQs).

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: the *Offline Clustering & Explanation* and the *Online Root Cause Diagnosis & Update*. 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:

1. We identify the problem of Intermittent Slow Queries in large-scale cloud databases, and design a scalable framework called iSQUAD that provides efficient and accurate root cause diagnosis for iSQs. It adopts machine learning techniques, while overcomes the inherent obstacles in terms of versatility, labeling overhead and interpretability.
2. We apply Anomaly Extraction of KPIs in place of boolean anomaly detection to distinguish anomaly types. A novel clustering algorithm TOPIC is proposed to reduce the labeling overheads for DBAs.
3. 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.
4. We conduct extensive experiments for iSQUAD's evaluation and demonstrate that our method achieves an average accuracy of 88%, *i.e.*, 40% 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 eight root causes of several hundred iSQs in two hours, which is approximately twenty times faster than traditional case-by-case diagnosis.

The rest of this paper is organized as follows: §2 describes iSQs and the 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 large-scale cloud database and our future work. §7 reviews the related work, and §8 concludes the paper.

2. BACKGROUND AND CHALLENGES

In this section, we first introduce background on iSQs in §2.1. Then, we present three key challenges in diagnosing the root causes of iSQs in §2.2.

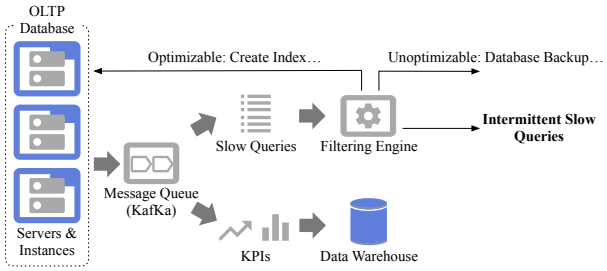
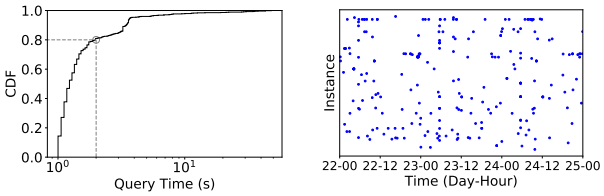


Figure 2: The architecture of the data collection system for Alibaba OLTP Database.



(a) CDF of Query Time (b) Scatter Plot

Figure 3: (a) CDF of the query time of iSQs. (b) Scatter plot of iSQs running on various instances over time.

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[1], Tmall[2], DingTalk[3], Cainiao[4], *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 [10] that continuously collects logs and KPIs (Key Performance Indicators). As shown in Fig. 2, in this work we focus on one common type of logs: those of slow queries.

Slow Queries: Slow queries can significantly degrade database performance when they occur. Most database systems, such as MySQL, Oracle, PostgreSQL, automatically log detailed information about queries whose query time is over a user-defined threshold [8, 35, 40]. The query time is the time lag between when an SQL query is submitted to, and when its results are returned by, the database. From the Alibaba Database, we obtain logs of all slow queries whose query time is more than one second. Alibaba's Filtering Engine classifies slow queries into three categories:

- *Optimizable slow queries* are those caused by poorly written SQL queries (*e.g.*, "bad" ordering of joins) or poor physical database designs (*e.g.*, non-optimal indexing choices). They account for roughly 20% of slow queries. For such queries, DBAs can often offer concrete suggestions or instructions for optimizing their performances.
- *Unoptimizable slow queries* are those for major database-wide events such as backing up databases or dumping history tables to database instances. They account for 79% of slow queries. They are fundamentally slow, and no optimization is possible.
- ***Intermittent slow queries*** are those whose facsimiles or similar queries are not slow in history. More specifically, queries with the same "features" as the iSQs' are not slow. Such features usually include template, logical read, execution plan, executed instances and physical machines. The iSQs, tens of thousands in number in our measurement dataset, account for 1% of the slow queries.

Dealing with iSQs is of great importance, since, when they occur unexpectedly, the experience of end users will be severely affected. iSQs might be more hazardous than that of other slow queries, though they are small in population. It is because optimizable slow queries have concrete solutions, unoptimizable slow queries are usually non-interactive and tolerable, but iSQs are unexpectable. Therefore, it is critical to design a solution to diagnosing the root causes of iSQs.

Fig. 3(a) shows the query time distribution of iSQs from Alibaba Database. As highlighted in Fig. 3(a), roughly 20% of iSQs take more than two seconds to execute. Fig. 3(b) plots the occurrences of iSQs over time (x-axis) across different instances (y-axis). As shown in Fig. 3(b), these occurrences appear to be scattered randomly over both time and "space" (y-axis), which suggests lack of temporal and "spatial" correlation among them. Hence we analyze their KPI behaviors to gain more insights.

KPIs: A Key Performance Indicator captures a system unit's real-time performance or how well it behaves in a database system. KPIs are one of the most important and useful monitoring data for DBAs in diagnosing performance issues. For example, KPI TCP Response Time (*tcp-rt*) is used in [10] to detect performance anomalies. Any single KPI alone, however, cannot capture all types of performance issues [33]. Indeed, many different KPIs are tracking various aspects of system running status. For instance, in MySQL, it is easy to configure hundreds of KPIs to be monitored [3]. Nonetheless, though we can monitor KPIs of various types, it remains challenging to use them to diagnose specific performance issues because KPIs are often interrelated and dependent. For example, when a performance issue, such as I/O saturation in a database instance, arises, I/O related KPIs and CPU utilization are all abnormally high at the same time; many other otherwise unrelated KPIs could become anomalous due to rapid fault propagation in database system, which exacerbates this situation.

In this work, we focus on the performance issues that cause iSQs. For each iSQ, we obtain the exact time duration and the locations (instances or physical machines) of the performance issues. With the help of experienced DBAs, we choose 62 KPIs, classified into 8 types as shown in Table 1, that cover almost all conceivable features of performance issues that may cause iSQs in real-life scenarios.

2.2 Challenges

We encounter three main 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. 1 as an example. Recall that the iSQs in Instance

[1] Taobao is a customer-to-customer online retail service.
[2] Tmall is a business-to-consumer online retail service.
[3] DingTalk is an enterprise collaboration service.
[4] Cainiao is a logistics service.

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

	Type	# KPIs	Example
Instance (48)	CPU	2	*docker.cpu-usage*
	I/O	18	*mysql.io-bytes*
	Workload	13	*mysql.tps*
	TCP RT [10]	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*

2, Case 1, which demonstrate a level shift-down in an I/O-related timeline and a spike-up in a CPU-related timeline, are caused by the I/O saturation in a physical machine. In contrast, the iSQs in Instance 2, Case 2, which display spike-ups in both I/O- and CPU-related timelines, result from a workload spike. If we deploy a conventional anomaly detection algorithm, observing anomalies on the same KPIs, we may come to the incorrect conclusion that the two groups of iSQs have the same root causes (while they actually do not). Furthermore, it is preferable to have a method that can achieve high accuracy, running time, and high scalability in detecting anomalies in large datasets.

Limitation of existing solutions: Different combinations of anomaly types may correspond to different root causes. Current anomaly detectors generally overlook the types of anomalies and over-generalize 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 exist strong correspondences and correlations among KPIs' anomalous performances and their root causes [6, 44], 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 profound domain knowledge, the labeling process is onerous [29]. For each KPI anomaly diagnosis, a DBA must first locate and log into a physical machine, and then inspect the anomaly-related logs and KPIs to reach a diagnostic conclusion based on them. 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. It is because we are going to standardize the results as our "diagnostic manual", on which all upcoming diagnoses will rely.

Limitation of existing solutions: Some previous works [44] 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 an enormous number of machines, instances, activities, interactions, *etc.* On the other hand, datasets of custom workloads are usually not in good conditions in terms of their 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 exorbitant 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 performances, 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 [44] 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 OF iSQUAD

This section introduces the framework of iSQUAD, which aims to diagnose the root causes of iSQs (§3.1), as shown in Fig. 4. We explain why we design the key components in our scenario in §3.2.

3.1 The framework of iSQUAD

The iSQUAD framework consists of two stages: the Offline Analysis & Explanation and the 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 KPI statistics have the same root causes. Therefore, it is a necessity that the model should draw the interconnection between iSQs and their root causes. Human DBAs may participate to elucidate this interconnection with high accuracy because of their domain knowledge. It is not possible to directly assign root causes to iSQ clusters without human labeling in advance. Thus, the offline part is primarily for clustering iSQs according to certain standard and presenting them to our human DBAs, so that DBAs can more easily recognize and correctly label the root causes. We feed the datasets of historical iSQs to the offline part, and then we concentrate on certain intervals given specific timestamps. Now, things are relatively straight-forward as we need to focus on only selected time intervals from KPIs' timelines and undertake anomaly extraction on KPIs within the intervals. After that, we have all anomalous KPIs in a discretized manner. Then, we apply the dependency cleansing on this intermediate result. Namely, if we have two abnormal KPIs A and B, and we have

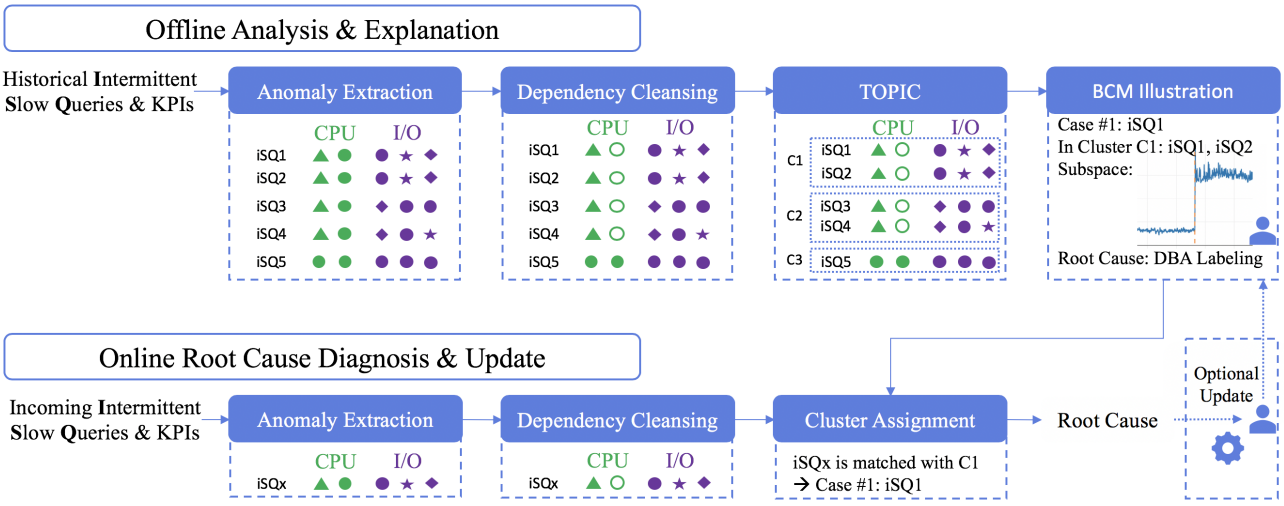


Figure 4: Framework of iSQUAD.

certain domain knowledge that A's anomaly tends to trigger that of B, we "cleanse" the anomaly alert on B. Thus, 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, and 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. Eventually, we present these clusters along with their corresponding representations to DBAs who are in charge of investigating and assigning root causes of every cluster of iSQs.

In the Online Root Cause Diagnosis & Update part, iSQUAD automatically analyzes a new online iSQ and its KPIs. We execute the online anomaly extraction and dependency cleansing as what we do in the offline part and gain its abnormal KPIs. Subsequently, we match the query to a cluster. Specifically, we compare this query with every clusters based on the similarity we define, and then match this query with the cluster whose pattern is the closest to this query's. After that, we use the root cause determined for this cluster by DBAs to help explain what triggers this iSQ. If the query is not matched with any existing clusters, a new cluster will be generated and DBAs will investigate and assign a root cause to it. New discovery in the online part can refine the ground truth given by the offline part.

3.2 Key Components

Anomaly Extraction: Recall that iSQs' own attributes, *e.g.*, timestamps, examined rows, are not fulfilling to convey suggestive information and thus cannot be optimized by the Filtering Engine. Consequently, we rely on the queries' KPI statistics. If we dive into the real world and understand how human DBAs check KPIs and diagnose anomalies, we may observe that experienced DBAs not only check the value of one KPI statistic (just like what people would assume), but also pay more attention to the state of it, *i.e.*, normal or of one of the anomaly categories like spikes and level shifts. This observation is the reason why we utilize Anomaly Extraction on each single KPI timeline. By doing so, DBAs are able to quickly distinguish multiple categories of anomalies.

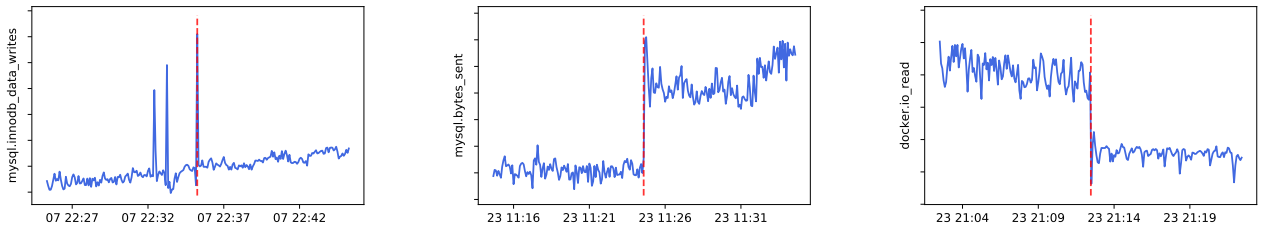
Dependency Cleansing: Many anomalies or problems lying in databases are "contagious": one faulty part can propagate the "disease" onto its related or neighboring parts. In another word, since some KPIs are highly correlated with each other, one anomalous KPI may be most of the time accompanied by another one or more anomalous KPIs. Analogous to the fact that a contagion can be either unidirectional or bidirectional, the relation between two KPIs is not necessarily mutual. Based on our domain knowledge, anomalies on instances are highly possible to incur their anomalous counterparts on physical machines, whereas problematic KPIs on physical machines may not always see corresponding problems on their instances. Therefore, we generally pick anomalous KPIs of instances using the measure of *confidence* [1], when the pairs of instances and physical machines are present, to guarantee independency. Results of this procedure are double-checked by DBAs.

Type-Oriented Pattern Integration Clustering (TOPIC): We have an incredibly huge number of iSQs but, as a common sense, the real root causes are actually limited in population [6]. Thus, this problem evolves to how to map those iSQs into a finite number of categories of root causes. Hence, we propose Type-Oriented Pattern Integration Clustering (TOPIC), an approach to clustering queries based on anomaly patterns as well as KPI types. It novelly encapsulates the gist of pattern matching, merging and similarity measuring. Different from DBSCAN [14], a representative of clustering algorithms that do not need a preset number of clusters, TOPIC can make use of the intricacies and dynamics of data patterns and types. Moreover, DBSCAN relies on distances among all the elements which have to be calculated individually, whereas our algorithm generalizes this problem and solves it more efficiently.

Bayesian Case Model: Clustering results alone are not interpretable enough for identifying all root causes of iSQs because iSQ clusters themselves convey little case-specific information. Therefore, the Bayesian Case Model (BCM) comes to light to extract the "meanings" of the clusters. BCM obtains the cases and representative features, and then forms human-understandable case-subspace expressions to highlight the key features of clusters, *i.e.*, a quintessential case and related anomalous KPIs for each cluster in our case. As a result, human DBAs can employ these pieces of information to more quickly and correctly find the root causes of iSQs.

4. DETAILED DESIGN

4.1 Offline Analysis and Explanation



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

Figure 5: Three common types of anomalies. Each red dash line signals the occurrence of an iSQ. (The values of KPIs are hidden for confidential reasons.)

In this section, we introduce the offline part of iSQUAD, which comprises Anomaly Extraction, Dependency Cleansing, Type-Oriented Pattern Integration Clustering and Bayesian Case Model.

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. 2). This type of KPI timelines or curves, however, can be very frustrating if a DBA is asked to directly stare at them and draw useful conclusions. As previously discussed, we extract anomalies from the KPIs and discretize the result for future use. Since whether a KPI is anomalous or not is insufficient to capture the patterns of iSQs, we not only detect KPI anomalies, but also identify the category for each KPI anomaly. For example, we determine whether a given anomaly is a spike, level shift-up, level shift-down (corresponding to Part (a), (b), (c) in Fig. 8 respectively) or even void (data missing). We catch this precious information as it can be exceptionally useful for query categorization and interpretation.

To identify spikes, we apply Robust Threshold [10] 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 via that time and generate two windows. Next, we examine whether the distributions of the two timelines are alike or not. If between the two a significant discrepancy is present and discovered by T-Test [37], 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 must ensure that all the KPIs we choose for consideration are independent from each other, so that no correlation or over-representation of KPIs impacts our result. To cleanse all the 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. According to the definition, we can get $confidence(A \rightarrow B) = \frac{|A \cap B|}{|A|}$, 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's value spans from 0 to 1, with the left extreme suggesting complete independence of two KPIs and the right extreme complete dependence. In our scenario, not only 1 denotes dependence. Instead, within the interval, we set a threshold, above which two KPIs are considered dependent to reflect real-world scenarios. We permute all KPIs and apply this strategy to each KPI pair. For example, an anomaly in some 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 very large extent. If we calculate the confidence, we might get the result "1", which suggests that these 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).

We search for an algorithm that incorporates the idea of "surjection", *i.e.*, able to cluster a large number of iSQs into a small set of clusters for root cause diagnoses. The problem is then transformed into how to properly define the similarities among elements, while considering specific conditions and how to separate and group the elements, so that the resulting clusters make sense or are inter-

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.

pretable at least. In our case, to define the similarities, we must consider both the patterns of iSQs and the different types of KPIs. It is a common sense that the more similar two queries are in terms of patterns, the closer they should stay, so that they are more likely to be grouped into the same cluster.

To calculate the similarity of the patterns of iSQs, the Simple Matching Coefficient [43], which computes two elements' percentage similarity in a bitwise fashion, comes to place very naturally as we have a huge number of discrete KPI states (normal or belonging to some anomaly category) to process for each iSQ pair. The relatively overwhelming number of indicators in certain types of KPIs, however, may unreasonably 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, the threshold σ, is used to determine how similar two iSQs need to 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.

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 so similar (how to properly 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

Algorithm 1: Type-Oriented Pattern Integration Clustering

Data: Intermittent slow queries under clustering
`S` $\leftarrow [iSQ_{index} : pattern]$
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)
```

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.

4.1.4 *Bayesian Case Model*

With results of TOPIC, we now aim to extract useful and suggestive information. Based on a large number of interviews to 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 we find quite suitable for this scenario in a general way. BCM is an excellent framework for extracting prototypical cases and generating corresponding feature subspace. Preserving high accuracy, BCM's case-subspace representation is 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. Taking Fig. 9 as an example to illustrate, for the shown cluster, BCM picks a specific iSQ (with Case ID 20190523118) and displays its fundamental KPIs that render it the prototypical case of this cluster.

BCM has some specifications that need to be strictly followed. First of all, 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.

4.2 Online Root Cause Diagnosis

By analogy to the offline segment, we follow the same procedures of the anomaly extraction and dependency cleansing to prepare the data for clustering analyses.

After receiving the discretized and cleansed patterns of a new iSQ, we are to match this query with a cluster for root cause diagnosis. Basically, we traverse the existing clusters' patterns and find one that is exactly the same as that of this incoming query or one that at rock bottom shares a high enough similarity score. If we indeed find one cluster that meets the requirement above, we match the query to this cluster and the root cause of that cluster logically explains this anomalous query. Otherwise, a new cluster is created for this "founding" query and DBAs are summoned 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.

4.3 Wrap Up the Framework

The framework of iSQUAD is a comprehensive root cause diagnosis tool for iSQs. The proper isolation of the offline and online sections guarantees that the online root cause diagnoses are accurate and reliable. It is because they are built on thoroughly investigated results by a number of experienced DBAs from a sophisticated cloud database service provider. Moreover, the scalable design allows the diagnoser to become increasingly "smarter" and inclusive, as certain diagnoses are taken as brand-new knowledge absorbed with DBAs' aids to continuously enhance the framework.

Table 2: Eight types of root causes of iSQs labeled by experienced DBAs. (Root Cause 4 - 6: accompanying queries' considerable actions including insertions, deletions, and update normally plunder excessive amount of I/O resources. Consequently, queries that share resources with these accompanying queries become iSQs.)

	Root Cause
1	CPU saturation in physical machine
2	CPU saturation in instance
3	I/O saturation in physical machine
4	Accompanying SQL's insertions
5	Accompanying SQL's deletions
6	Accompanying SQL's updates
7	Workload spike
8	Network congestion

5. EVALUATION

In this section, we describe in details how we initialize and conduct our experiments to assess iSQUAD and its major components. We set up the experiments in §5.1, and individually evaluate the whole iSQUAD in §5.2, Anomaly Extraction in §5.3, Dependency Cleansing in §5.4, TOPIC in §5.5, and BCM in §5.6, respectively.

5.1 Setup

Datasets of Intermittent Slow Queries: We use a large number of real-world iSQs, collected from diverse workloads of Alibaba OLTP Database, as our datasets in the study. Specifically, DBAs at Alibaba use their Filtering Engine to categorize slow queries on instances into the optimizable, unoptimizable, and the third type that stands for the iSQs. We first randomly select a certain number of instances running on the physical machines of Alibaba Database which is simultaneously executed on millions of physical machines. Then, for each instance, we select one iSQ at each specific timestamp. That is, we select only one iSQ for each unique instance-host-timestamp signature. This selection policy is because we can safely assume that choosing one for analysis is sufficient to represent its group of homogeneous queries. In total, we obtain three datasets of iSQs, one for an arbitrary day and two for one week each. These datasets are random and complete for investigation. In this way, our analyzed iSQs can 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. 2. 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 62 KPIs in total, which are carefully selected by DBAs from hundreds of KPIs as the representatives.

Ground Truth: We ask a dozen of experienced DBAs from Alibaba to label the ground truth for evaluating each component of iSQUAD and the whole iSQUAD. Considering the sheer size of our datasets, we randomly sample the datasets for labeling purpose. To fully evaluate each component of our framework, the

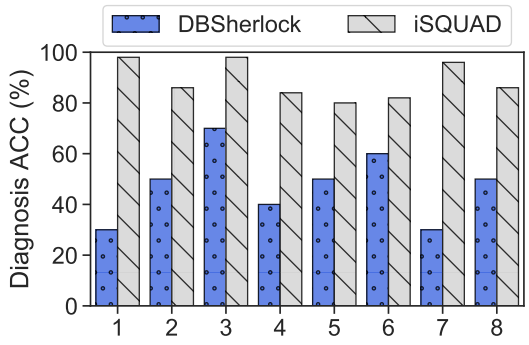


Figure 7: Diagnosis accuracy (ACC) of DBSherlock [44] and iSQUAD in online diagnoses of the eight types of root causes.

set of ground truth includes the parts for Anomaly Extraction, Dependency Cleansing, Type-Oriented Pattern Integration Clustering, Bayesian Case Model, and iSQUAD as a whole, respectively. For Anomaly Extraction, DBAs label anomaly types of KPIs from an arbitrary day. For Dependency Cleansing, DBAs analyze the aforementioned 62 KPIs and discover ten underlying dependencies among them. For both TOPIC and BCM, DBAs carefully label 175 iSQs with root causes. For the iSQUAD framework, each one of the root causes shown in Table 2 has ten sampled iSQs. This ground truth set is considerable in size because of labeling overhead and comparable with what was used in DBSherlock [44]. The details of manual labeling are as follows: experienced DBAs spend one week analyzing one hundred of iSQs one by one random chose from three datasets. They find that the root causes of iSQs can be summarized as eight types as shown in Table 2. The occurrences of different types of iSQs are almost equally likely, so they random pick ten iSQs that correctly belong to each one of the root causes. This setting is comparable with that of DBSherlock, and is not biased 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 Diagnosis Accuracy, Clustering Accuracy, NMI, and F1-score. More details of these metrics are presented as follows.

Diagnosis Accuracy: measures to what extent the root cause diagnoses are correct and is computed by $\frac{\#Correct\ Diagnosis}{\#All\ Diagnosis}$. It is used to measure iSQUAD performance in §5.2.

F1-Score [38]: is the harmonic mean of the precision and recall. F1-score is used to measure the performance of Anomaly Extraction (§5.3) and Dependency Cleansing (§5.4).

Clustering Accuracy [46]: is to find the bijective maps between the clusters and the ground truth classes, and then measure to what extent each cluster contains the same data as its corresponding class does. (§5.5)

Normalized Mutual Information (NMI) [9]: quantifies the "amount of information" obtained about one cluster through observing another cluster, and is a good measure of the clustering quality. (§5.5)

Implementation Specifications: Our framework of iSQUAD is implemented using Python 3.6 and our study is conducted on a Dell PowerEdge R420 server with an Intel Xeon E5-2420 CPU and a 24GB memory.

5.2 iSQUAD Accuracy & Efficiency

As mentioned in §5.1, our sample consists of ten iSQs for each specific root cause which can explain a iSQ cluster. We pick one iSQ as the starter of a cluster and then evaluate this cluster by comparing the 9 remaining iSQs with it and computing the accuracy. To reduce potential outlier impacts, we repeat the process above ten times by rotating the starter iSQ. Finally, we calculate their average accuracy. As shown in Fig. 7, the accuracy of iSQUAD (about 88%) is much higher than that of DBSherlock (about 40%). Moreover, the computation time of iSQUAD is 0.79 second per cluster while that of DBSherlock is 1.49 second per cluster. This demonstrates that iSQUAD outperforms DBSherlock in terms of both the accuracy and efficiency.

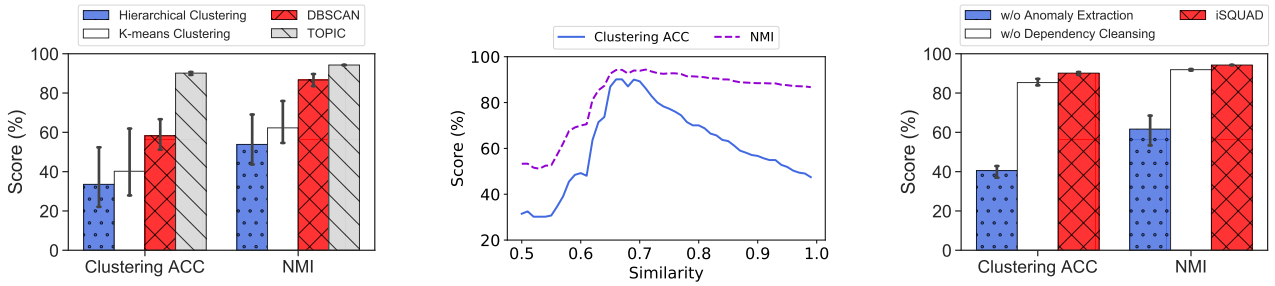
iSQUAD performs better than DBSherlock in four aspects. 1) DBSherlock requires user-defined or auto-generated abnormal and normal intervals of a KPI timeline. This requirement deviates from that only exact timestamps are provided here. DBSherlock's algorithm may not produce satisfactory predicate combinations 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 seemingly feasible, but actually chokes up 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 vivid and understandable case-subspace representations as shown in Fig. 9 to DBAs. To sum up, iSQUAD is impressively interpretable with high accuracy.

Table 3: Performance comparison of popular anomaly detection methods.

	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 [31, 45]	60.7	6.06

5.3 Anomaly Extraction Accuracy & Efficiency

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 [31, 45], and the results are presented in Table 3. Both dSPOT and iSST are representatives of state-of-the-art spike and



(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 on three datasets under different similarity requirements. (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 original and complete iSQUAD we propose. (The shown results are the best scores generated by tuning parameters. Black error bars indicate the diversity and variance in our three datasets.)

Table 4: Performance comparison of dependency measures.

Method	Precision (%)	Recall (%)	F1-Score (%)
Confidence	90.91	100	95.24
MI [44]	100	40	57.14
Gain Ratio [20]	87.5	70	77.78

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. Comparatively, 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 4. 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], DBSCAN [14]) in our scenario. The metrics 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 a preferable choice because it takes both KPI types and anomaly patterns into account. This mechanism is intuitive for clustering iSQs with multiple KPIs. We hereby analyze the reasons why the three traditional clustering algorithms (hierarchical clustering, K-means, and DBSCAN) are not as excellent 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

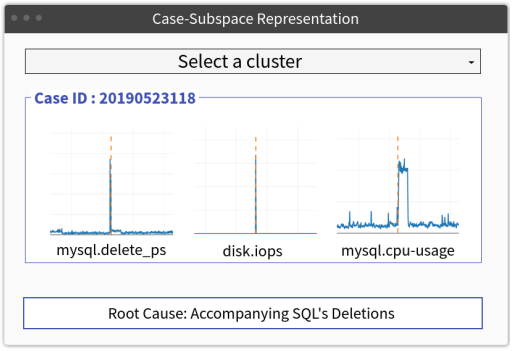


Figure 9: Visualization platform. (Exact values are hidden for confidential reasons.)

of iSQs.

Parameter Sensitivity: In our clustering method, TOPIC, the similarity percentage is one crucial threshold that describes to what level two KPIs' patterns can be considered similar enough to get integrated into one. This threshold consequently influences directly the shapes and number of final clusters. We further investigate the impact of the similarity threshold, whose results are shown in Fig. 8(b). (The shown values are the average of results of three datasets.) From this figure, we observe that the performance trends of the accuracy and NMI are similar. 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 0.68. After the similarity score of 0.7, both of them begin to diverge. Both the accuracy and NMI drop while the former plunges even more because as the similarity requirement gets overly strict, some very similar iSQs that are supposed to be together are forced to be segregated irrationally. Therefore, as the similarity increases too much, the number of iSQs per cluster drops and the clustering accuracy decreases.

Positive Effects of Previous Components on TOPIC: We further 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,

Table 5: 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

DBAs spend about two hours diagnosing eight cases produced by iSQUAD. On the contrary, they take one week (40 hours) to analyze all iSQs in the traditional way, *i.e.*, case-by-case diagnosis without BCM. Therefore, diagnosing root causes of iSQs using iSQUAD can be twenty times faster than the traditional diagnosis.

Visualization Platform: The Bayesian Case Model is embedded in the visualization platform (Fig. 9) 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.

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 5. In particular, this performance improvement is more significantly shown by DBAs who have advanced database knowledge.

6. CASE STUDY AND FUTURE WORK

Case Study: A prototype of iSQUAD has been successfully deployed to Alibaba Database. As illustrated in Fig. 2, Alibaba has originally implemented a configuration to partially reduce the negative effects of slow queries. Its basic mechanism is to extract slow queries from the Message Queue, and then feed them into the Filtering Engine that is responsible for slow query categorization. Before iSQUAD, Alibaba has managed to deal with both optimizable slow queries and unoptimizable slow queries, but does not have any method to help handle iSQs. Now, iSQUAD is smoothly integrated into Alibaba Database and is sophisticated enough with iSQ diagnoses. Therefore, this architecture is currently better-rounded. It is able to surveil all the slow queries occurred inside the database and then handle them differently with respect to their categories.

Take a real-life iSQ diagnosis event as an example. At 19:07, 17^{th} June, 2019, an "communications link failure" alert is triggered and on-call DBAs are called to work on it because this failure has significantly put the brakes on queries' executions. DBAs check the iSQ processed by iSQUAD at the alert time. Along with this query, iSQUAD also quickly generates and displays the root cause diagnosis to the DBAs. Precisely, the anomalous and prominent KPIs are abnormally high in *mysql.delete-ps, disk-iops* and *mysql.cpu-usage*. In this case, iSQUAD's diagnostic root cause is "Accompanying SQL's deletions", which means a large number of data deletions simultaneously cause I/O jams, slowing down other SQLs that are supposed to be fast from historical statistics. As a solution, DBAs put a rate-limiting threshold and subsequently notice

that the alert is dismissed. This example clearly demonstrates that iSQUAD is a practical tool that is able to help DBAs diagnose the root causes of iSQs. Remarkably, the whole process takes less than ten minutes while this type of tasks traditionally take 3.5 hours for each on average. It saves human's time by recommending a very accurate root cause of an iSQ, so that DBAs can jump directly into the cause-removal step, instead of searching various KPIs around to figure out a root cause like what they normally do.

Future Work: On one hand, the application of iSQUAD is not strictly limited to iSQ root cause diagnosis only. In our case, iSQs provide iSQUAD with detailed information like timestamps, instance IDs, and physical machine IDs. These concise but valuable information pieces can precisely guide iSQUAD to complicated root causes. By analogy, we speculate that iSQUAD is also capable of diagnosing a variety of root causes of anomalies given specific timestamps and venues - not only in databases but also in many other scenarios. We plan to examine our speculation in the future with more diverse datasets.

On the other hand, to better facilitate the practical usage of iSQUAD, we recommend problem-solving procedures of tackling two main categories of iSQ root causes: 1) For the problems caused by other slow queries' considerable actions, *i.e.*, insertions, deletions, or updates, we recommend DBAs to apply rate-limiting thresholds onto the unoptimizable slow queries. 2) For the problems that occur on instances or physical machines, we suggest that DBAs redirect the workloads of the anomalous instances or physical machines to backup instances. Theoretically, iSQUAD does not end here. The aforementioned solutions may not be complete enough to handle all the root causes we mention in Table 2. For 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

We classify the previous works related to root cause diagnoses of iSQs into four categories: the slow query analysis, anomaly extraction, clustering algorithm, and anomaly explanation.

Slow Query Analysis: Slow query analysis and optimization [7] have been extensively studied. General approaches involve data-driven automatic analyses and optimizations of databases and queries. For databases, several previous studies [13, 27, 39] aim to automate indexing modifications to achieve better performance, and one study addresses the issue of tuning database parameters with machine learning algorithms [42]. On the side of query optimization, boosting queries by deep learning is introduced in [24, 32], and slow queries in the Storage Area Network are analyzed in [7]. Neither of the above, however, touches the field of iSQs. Our work is the first to reduce negative effects of iSQs on database systems by proposing iSQUAD.

Anomaly Extraction: Previous anomaly detection algorithms generally output binary results *i.e.*, either "normal" or "anomalous". In the literature, there are a variety of anomaly detectors, such as Opprentice [29], dSPOT [41] and iSST [31, 45]. Also, in industry, some corporations develop several anomaly detectors, *e.g.*, Yahoo's EGADS [26], 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 different insights. The K-Shape clustering [36], built on [5], is to cluster queries based on KPI timelines' shapes. This proposition is off from our scenario because we focus on one timestamp across all KPIs while K-Shape clustering allows a time lag between two similar shapes. Such a latency may cluster two irrelevant queries together and incur accuracy loss. Next, the workload compression technique in [30] is similar to our work. It calculates the similarity of the workload features based on the cosine similarity. One drawback is that it loses the information of distinct KPI types, which play important roles in determining query behaviors. By contrast, our TOPIC coalesces the effects of both KPI types and anomaly patterns to cluster queries in a pragmatic and rigorous manner. Moreover, TOPIC does not modify the cluster centers, *i.e.*, anomaly patterns, of existing clusters like [30] does. This design is because patterns, which are integrated when being merged, are stable unlike templates in [30] 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 which require relevant identified task pairs. The predicate-based explanations of PerfXplain are similar to those of DBSherlock [44], 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 large-scale cloud databases. Countless 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. Based on Anomaly Extraction, Dependency Cleansing, Type-Oriented Pattern Integration Clustering, and Bayesian Case Model, iSQUAD can, to a very large extent, 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 all 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. REFERENCES

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

[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 Conference*, pages 440–453. ACM, 2016.

[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. *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 *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 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 5th European conference on Computer systems*, pages 111–124. ACM, 2010.

[7] N. Borisov, S. Babu, S. Uttamchandani, R. Routray, and A. Singh. Why did my query slow down? *arXiv preprint arXiv:0907.3183*, 2009.

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

[9] 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.

[10] 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 International Conference on Management of Data*, pages 615–627. ACM, 2018.

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

[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 SIGMOD*. ACM, 2019.

[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 *Kdd*, volume 96, pages 226–231, 1996.

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

[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 International Conference on Management of Data*, pages 333–348. ACM, 2019.

[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. *International Journal of Information Technology and Knowledge Management*, 2(2):271–277, 2010.

[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. *Proceedings of the VLDB Endowment*, 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 *Advances in Neural Information Processing Systems*, pages 1952–1960, 2014.

[24] 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. 2019.

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

[26] N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th SIGKDD*, pages 1939–1947. ACM, 2015.

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

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

[29] 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 Internet Measurement Conference*, pages 211–224. ACM, 2015.

[30] 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 International Conference on Management of Data*, pages 631–645. ACM, 2018.

[31] M. Ma, S. Zhang, D. Pei, X. Huang, and H. Dai. Robust and rapid adaption for concept drift in software system anomaly detection. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 13–24. IEEE, 2018.

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

[33] 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.

[34] 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 international conference on management of data*, pages 301–312. ACM, 2013.

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

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

[37] 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.

[38] D. M. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

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

[40] H.-J. Schnig. 3 ways to detect slow queries in postgresql. `https://www.cybertec-postgresql.com/en/3-ways-to-detect-slow-queries-in-postgresql/`, 2018.

[41] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd SIGKDD*, pages 1067–1075. ACM, 2017.

[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 International Conference on Management of Data*, pages 1009–1024. ACM, 2017.

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

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

[45] 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 11th CONEXT*, page 2. ACM, 2015.

[46] N. Zhao, L. Zhang, B. Du, Q. Zhang, J. You, and D. Tao. Robust dual clustering with adaptive manifold regularization. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2498–2509, 2017.