DBCatcher: A Cloud Database Online Anomaly Detection System based on Indicator Correlation

Guangyu Zhang[1], Chunhua Li[1*], Ke Zhou[1], Li Liu[1], Ce Zhang[2], Wancheng Chen[1]
Haotian Fang[1], Bin Cheng[3], Jie Yang[3], and Jiashu Xing[3]
[1] WNLO, Huazhong University of Science and Technology, China
[2] ETH Zurich, Switzerland
[3] Tencent Inc, China
{zhangguangyu, li.chunhua, zhke, lillian_hust, chenwc, fanght}@hust.edu.cn
{ce.zhang}@inf.ethz.ch;{bencheng, edgeyang, flacroxing}@tencent.com

***Abstract*—Anomaly detection system plays an important role in maintaining the stability of cloud database. Existing studies mainly focus on significant deviations in multivariate time series, such as a combination of CPU utilization, transactions per second, etc, to detect abnormal issues. Due to the complexity of cloud database structure and functions, these approaches are difficult to achieve a balance among detection performance, detection efficiency and workload adaptability. In this paper, we propose DBCatcher, a cloud database online anomaly detection system based on indicator correlation. Through extensive analysis of real-world cloud database time series, we find the correlations among trends in the same key performance indicators across databases within the same unit, which inspires us to explore a time series correlation measurement method that can efficiently detect abnormal issues. Meanwhile, we design a flexible time window observation mechanism and an adaptive threshold learning policy to minimize misjudgment caused by key performance indicator fluctuations, greatly enhancing the detection performance and workload adaptability. We conduct extensive experiments under real-world and synthetic workloads. Experimental results show that DBCatcher significantly improves the detection performance and detection efficiency compared to existing methods.**

***Index Terms*—anomaly detection, cloud database, time series, correlation measurement**

I. Introduction

Cloud database is growing rapidly in the area of infrastructure services provided by cloud service vendors (e.g., AWS Cloud, Microsoft Azure, and Tencent Cloud). Gartner forecasts a 38.2% annual growth rate in the cloud database market from 2021 to 2026 [1]. Therefore, maintaining the reliability of cloud database becomes an increasingly critical issue [2]. Experienced Database Administrators (DBAs) have found that when trends in the time series of Key Performance Indicators (KPIs), such as CPU utilization, transactions per second, deviate significantly from normal patterns, it commonly represents some abnormal issues in cloud databases [3], [4]. As a result, cloud service vendors establish monitoring systems to observe the changes in KPIs to determine the operational state of cloud database. The monitoring data on multiple KPIs constitutes a multivariate time series.

Existing studies on time series anomaly detection are classified into two main categories: statistical based methods [5]–[8] and machine-learning based methods [9]–[16]. To observe the effectiveness of mainstream anomaly detection methods on time series collected in practical cloud database products, we evaluated a few representative methods, including FFT [7], SR [8], SR-CNN [14], OmniAnomaly [15], and JumpStarter [16], on various time series collected by Tencent Cloud Database Monitoring System. FFT and SR are anomaly detection methods based on statistical knowledge, while SR-CNN, OmniAnomaly, and JumpStarter are machine-learning based methods. Table I shows the limitations of these methods in detection performance, detection efficiency, threshold auto-adjustment, and workload adaptability.

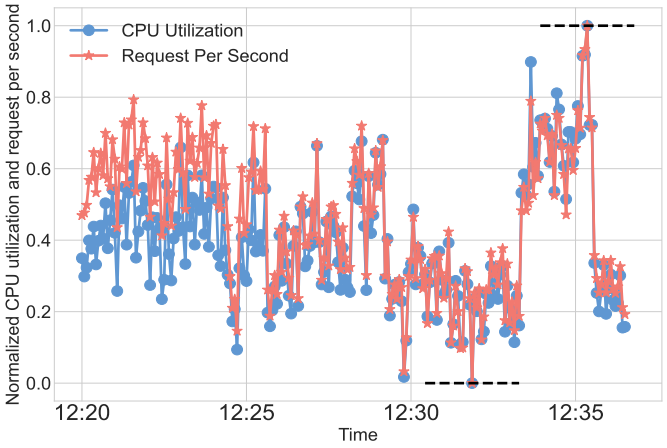


Fig. 1. An example of a burst increase in "CPU utilization" due to an increase in "requests per second", where "CPU utilization" and "requests per second" are normalized to show time-series trends. Such trends are universal in Tencent OLTP Cloud Database, for example, e-commerce users or game users will have a burst of requests at some point in time.

(1) Detection performance. Cloud database multivariate time series have various variation patterns. Some methods (e.g, FFT, SR) detect the anomalies of time series by setting deviation thresholds, while the difference in variation patterns of every KPI makes it difficult to set appropriate deviation thresholds, limiting their scope of applicability in practice.

(2) Detection efficiency. Detection efficiency means how long it takes for a detection method to find anomalies. Each cloud database processes tens of thousands of critical online trans-

TABLE I
CHARACTERISTICS OF DIFFERENT ANOMALY DETECTION METHODS. THE METHOD WE PROPOSE IS DBCATCHER.

	FFT	*SR*	*SR-CNN*	*OmniAnomaly*	*JumpStarter*	*DBCatcher*
Detection performance	Low	Low	Medium	High	High	High
Detection efficiency	Low	Medium	High	Medium	Medium	High
Threshold auto-adjustment	Low	Low	Low	Low	Low	High
Workload adaptability	Low	Low	Medium	Medium	Medium	High

actions per second. Therefore, detection efficiency is directly correlated to the quantity and scope of transactions affected by anomalies. Existing anomaly detection methods (e.g, OmniAnomaly, JumpStarter) require extensive data points (e.g., one data point per 5 seconds) to learn time series variation features, resulting in anomaly identification too slowly.

(3) Threshold auto-adjustment. Most existing studies require multiple thresholds for anomaly detection and these thresholds are frequently a crucial part of the ultimate evaluation of database state [3], [17]. Setting these thresholds under different workloads is non-trivial even for experienced DBAs, as inappropriate thresholds can have side effects on detection performance and efficiency.

(4) Workload adaptability. Existing methods are particularly susceptible to workload variations. When workload varies, the performance of existing machine-learning methods that have previously been trained can plummet (e.g, SR-CNN, OmniAnomaly, and JumpStarter) [16]. Statistical based methods (e.g, FFT, SR) can also be quite challenging under varied workloads because of the difficulty of threshold adjusting.

The conclusions in Table I indicate that the mainstream anomaly detection methods are not applicable to cloud databases. By carefully analyzing the multivariate time series in Tencent Cloud Database[1], we find the following differences between cloud database time series and the time series of mainstream anomaly detection service objects (e.g, online services): (1) Cloud database multivariate time series change more frequently and with greater magnitude than other online service time series. Figure 1 shows an example of the burst increase in "CPU utilization" due to the increase in "Request Per Second" increases in cloud database time series. (2) Cloud database time series have complex variation patterns. Other online service time series are commonly characterized by periodic variations in time granularity by day, hour, etc. [15], [18], [19], while cloud database time series contain extensive irregular time series. (3) Complex functionality of cloud databases such as data synchronization, consistency maintenance, etc, can cause complex abnormal issues [4], [20], [21]. These complex abnormal issues generate a wide variety of time series trends (e.g., concept drift, spike) [4], [22]. The above analysis and findings prompted us to investigate anomaly detection methods applicable to cloud databases.

In this paper, we find correlations among trends in the same KPIs across databases within the same unit (see §II-B for details). This phenomenon is called Unit Key Performance Indicator Correlation (UKPIC). Based on the UKPIC phenomenon, we propose an efficient cloud database online anomaly detection system called DBCatcher.

The key contributions in this paper are as follows:

- We propose an efficient time series correlation measurement method based on the UKPIC phenomenon to calculate the correlations of time series, which can timely identify KPIs with abnormal variation trends.
- We propose a flexible time window observation mechanism that considerably enhances the performance of cloud database anomaly detection by reducing the detrimental impact of temporal fluctuations in KPI changes.
- We propose an adaptive threshold learning policy that addresses the challenge of auto-adjusting thresholds under varying workloads.
- We conduct experiments under real-world and synthetic workloads. Experimental results show that DBCatcher improves F-Measure by 8.3%, 8.8%, and 9.2% over state-of-the-art methods, it also accelerates detection efficiency by 2.5× to 3×. For a 100M dataset, corresponding to the amount of data for 120 hours of KPI data points, DBcatcher takes only 42 seconds, which provides an acceptable time overhead for online detection.

II. BACKGROUND AND PRELIMINARY STUDY

In this section, we first give the necessary background and in-depth analysis of anomaly detection under cloud databases, including the architecture of cloud databases, the unit key performance indicator correlation phenomenon, and abnormal time series trends. Then we present the challenges of anomaly detection by applying correlation measurement methods.

A. Cloud Database Architecture

Figure 2 shows the general architecture of cloud databases. As shown in the figure, a database cluster contains multiple units, each unit deploys a load balance module and multiple databases, and each database has one primary instance and multiple replica instances. SQL requests from upper-level applications are distributed to different database units through the global transactions manager, which are further processed by the load balance module and forwarded to different databases in the same unit. For read requests, they are handled equally by different instances of each database. For write requests, they are first processed via the primary instance and then the data is synchronized to the remaining replica instances. When

[1]Tencent Inc. is the largest social network service company in China, which cloud databases provide data storage and management services for massive users, supporting various applications such as social networks, games, e-commerce, and finance.

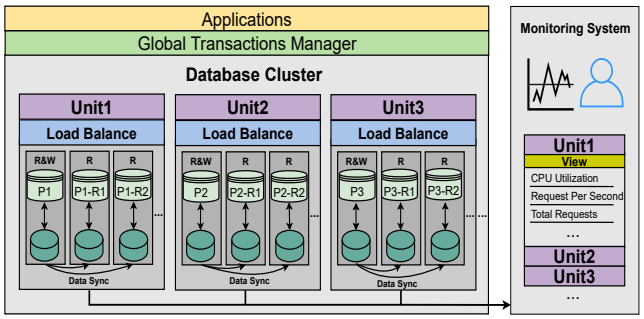


Fig. 2. Cloud database architecture and load balancing scenarios. In general, each geographical area has a database cluster, and a database cluster contains multiple units. A unit has multiple databases, each with a primary instance and multiple replica instances. Multiple instances of a database are distributed in different servers or racks for high availability.

a failover occurs, a replica instance is selected as the new primary instance and request processing continues as before. To observe the operational state of each database, cloud service vendors typically establish a bypass monitoring system that collects KPI time series of cloud databases at an equal time interval. DBAs determine whether a database is in a "healthy" or "abnormal" state by observing the time series trend of KPIs. This similar cloud database architecture is widely used by multiple cloud service vendors such as Amazon AWS [23], [24], Alibaba Cloud [2], and Tencent Cloud [25] etc.

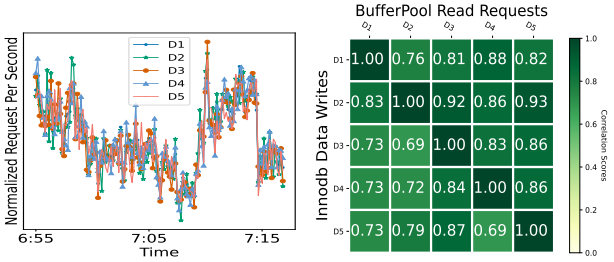
B. Unit Key Performance Indicator Correlation

Based on long-term observation to cloud database multivariate time series, we notice correlations in the trends of the same KPI time series in a unit. Figure 3(a) shows the trend of normalized "Request Per Second" of five databases in a unit. Although the value of "Request Per Second" varies from database to database at each point, the overall trend is correlated. In Figure 3(b), we have also summarized the correlation scores among different databases for "Innodb Data Writes" and "BufferPool Read Requests". We find strong correlations among the different databases within the same unit regarding the two KPIs. This phenomenon is called Unit Key Performance Indicators Correlation (UKPIC).

In Table II, we have summarized 14 KPIs with the UKPIC phenomenon in cloud databases. We classify correlations into two types according to the database usage: `P-R` and `R-R`. `P-R` indicates that the primary database has UKPIC phenomenon with replica databases, whereas `R-R` indicates that the replica database has UKPIC phenomenon with other replica databases. We have carried out an in-depth analysis of the UKPIC phenomenon and found that there are two main reasons for this phenomenon: (1) Due to the load balancing module, the number of SQLs processed by each database is similar. (2) Each database handles transactions with similar characteristics. Transactions in OLTP applications all have high real-time requirements and short processing time.

TABLE II
INDICATORS DESCRIPTION AND CORRELATION TYPE

Indicator Name	Correlation Type
Com Insert	R-R
Com Update	R-R
CPU Utilization	P-R, R-R
BufferPool Read Request	P-R, R-R
Innodb Data Writes	P-R, R-R
Innodb Data Written	P-R, R-R
Innodb Rows Deleted	R-R
Innodb Rows Inserted	R-R
Innodb Rows Read	P-R, R-R
Innodb Row Updated	P-R, R-R
Requests Per Second	P-R, R-R
Total Requests	P-R, R-R
Real Capacity	P-R, R-R
Transactions Per Second	R-R



(a) correlation change trend (b) correlation scores

Fig. 3. (a) the correlation trend regarding "Requests Per Second" of the five databases in a unit. (b) the upper triangular matrix shows the correlation scores of the databases regarding "BufferPool Read Requests", and the lower triangular matrix is the correlation scores regarding "Innodb Data Writes".

C. Abnormal Time Series Trends

It is rare for multiple databases to have abnormal issues at the same time, so this paper only considers the case of a single database has abnormal issues [26]. When an abnormal issue occurs in one of the databases in a unit, multivariate time series of that database will disrupt the UKPIC phenomenon. In Figure 4, we show an example of an abnormal issue in real cloud-database scenario. When a defective load balancing strategy is deployed, extensive SQL statements are centrally mapped to `D1`, causing several KPIs of `D1` to deviate compared to other databases.

Numerous studies have identified many abnormal types of time series trends such as concept drift, spike, and level shift, etc [4], [22], [27]. In cloud-database scenarios, these abnormal types appear as a common feature. The multivariate time series trends of abnormal databases deviate from the multivariate time series trends of other databases [28]. As a result, correlation measurement methods can aid in detecting database state. When the multivariate time series trends in a database are uncorrelated from those of other databases in the same unit, the database is likely to contain abnormal issues.

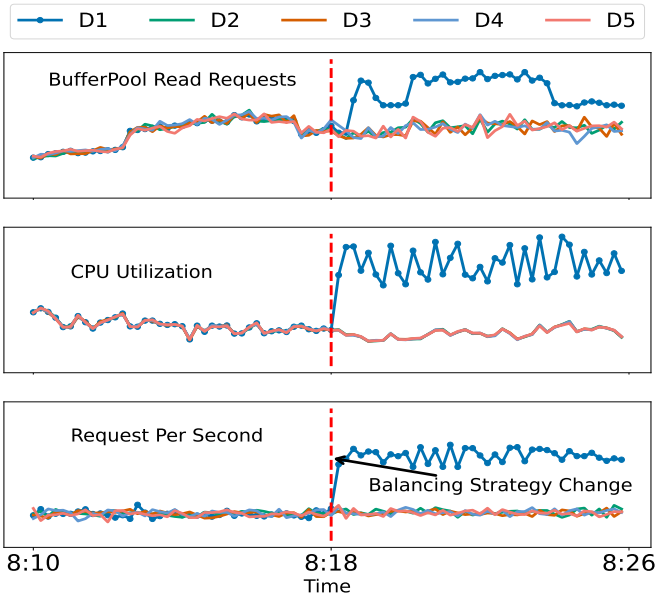


Fig. 4. The multivariate time series (selected as examples) of an unit before and after a balancing strategy change. The red vertical line followed by multivariate time series after the occurrence of a abnormal issue.

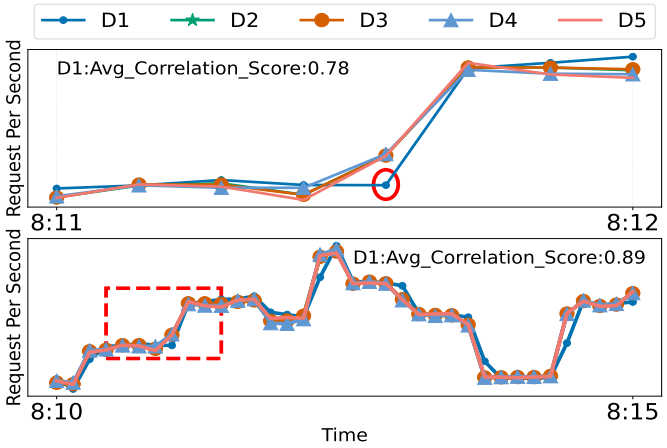


Fig. 5. Individual point temporal fluctuations within a shorter time interval have a significant impact on correlation scores. By expanding the time interval (e.g., 5 minutes), the effect of temporal fluctuations on the correlation score can be mitigated, but this affects the detection efficiency.

D. Challenges

Inspired by the UKPIC phenomenon, we expect to find KPIs with abnormal deviation trend through correlation measurement method, and thus to detect abnormal problems. However, we encounter three challenges.

Point-in-time delay. Correlation measurement methods are mostly used to judge the correlation between two time series. In cloud-database scenario, complex software composition may cause delays in data collection, processing, and distribution. These point-in-time delays result in an offset between the data points to be compared in the two time series. Pearson coefficient [29] doesn't take into account the delay among data points. Dynamic time warping [30] is tough to find the exact number of point-in-time delays. Therefore, it is challenging to obtain accurate correlation scores in time series with point-in-time delays.

Temporal fluctuations. In Figure 5, we show the effect of temporal fluctuations on the correlation scores. It is vital to note that the difference between temporal fluctuations and anomalies is that temporal fluctuations are minor deviations at individual points and gradually the time series return to normal changing trends. The causes of temporal fluctuations are as follows: (1) the presence of complex functions (e.g, maintenance tasks) in cloud database, can affect the variation trends of some KPIs (e.g., CPU utilization). (2) complex workloads make absolute load balancing tough to achieve. How to reduce the impact of temporal fluctuations on detection efficiency and detection performance is an important issue.

Threshold adjustment. More KPIs related to the UKPIC phenomenon may become available as load balancing strategy technology evolves. While the discovery of the UKPIC phenomenon has the potential to improve detection performance, it also brings some additional challenges. For multiple KPIs, we need to find multiple appropriate correlation thresholds to ultimately identify anomalies, which would increase artificial dependence. Furthermore, frequent changes in cloud database workloads force the detection system to adjust these thresholds on a regular basis. Otherwise, the detection performance of the system can degrade significantly.

III. DBCatcher

In this section, we first provide an overview of the proposed system DBCatcher. Then, we describe in detail three key technologies employed for the system: an efficient time series correlation measurement method, a flexible time window observation mechanism, and an adaptive threshold learning policy. These technologies can address the challenges (§II-D).

A. System Overview

Figure 6 shows the system overview of DBCatcher. DBCatcher consists of a data processing module, a correlation measurement module, a streaming detection module, and an online feedback module. The data processing module is in charge of collecting KPI time series of all cloud databases in a unit, with a collection interval of 5 seconds among data points. The data processing module maintains multiple queues for each KPI, the number of which is equal to the number of databases in the unit. The correlation measurement module extracts data points for each KPI and adds them to a time window. The correlation measurement module then adopts an efficient time series correlation measurement method (§III-B) to calculate the correlation scores of different databases on the same KPI time series. After that, the streaming detection module uses correlation scores to determine the database state through a flexible time window observation mechanism (§III-C). In the online feedback module, DBAs mark the database state judgment records obtained by the stream detection module. When workload changes cause the

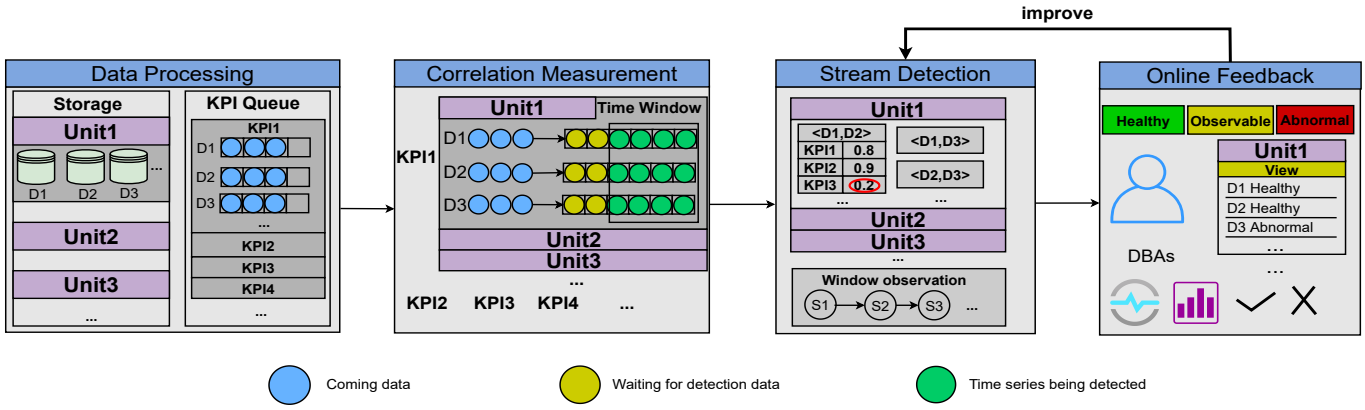


Fig. 6. The overview of DBCatcher. It consists of a data processing module, a correlation measurement module, a streaming detection module, and an online feedback module.

detection performance provided by the initial thresholds to fail to meet current system requirements, the online feedback module employs an adaptive threshold learning policy (§III-D) to adjust thresholds through recent judgment records.

B. Efficient Time Series Correlation Measurement

In this section, we describe how to resolve the issue that the presence of point-in-time delays in time series makes it hard to obtain accurate correlation scores. We first describe how to calculate the correlation scores of two time series with point-in-time delays. Then, we describe how to preserve the correlation relationship of multivariate time series.

We denote two time series as : $\vec{x} = (x_1, ..., x_n)$ and $\vec{y} = (y_1, ..., y_n)$, where n is the number of data points in a given time window. In cloud-database scenario, $\vec{x}$ and $\vec{y}$ represent the same KPI time series of two databases in a unit. We are concerned with the correlation relationship between $\vec{x}$ and $\vec{y}$ rather than specific values. Therefore, we normalize $\vec{x}$ and $\vec{y}$ in the following manner:

$$\vec{x} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{1}$$

where $i \in [1, n]$. Data collecting delays may cause phase offsets in the parts of the two time series that should be compared. The compared portion of $\vec{x}$ and $\vec{y}$ is denoted as:

$$correlation(\vec{x}, \vec{y}) = \begin{cases} (x_1, ..., x_s, \overbrace{x_{s+1}, ..., x_n}^{n-s}) \\ (\underbrace{y_1, ..., y_{n-s}}_{n-s}, y_{n-s+1}, ..., y_n) \end{cases} \tag{2}$$

where s is the possible delay length. The part that should be compared is $x \in [s+1, n]$, $y \in [1, n-s]$. Since the delay s is not known in advance, we need to traverse the possible delay lengths when calculating the correlation score. The possible correlation score for $\vec{x}$ and $\vec{y}$ is abbreviated as cs. cs is calculated as:

$$cs_s = \begin{cases} \sum_{i=1}^{n-s} (\vec{x}_{i+s} - ave(\vec{x})) \cdot (\vec{y}_i - ave(\vec{y})) & s \geq 0 \\ \sum_{i=1}^{n+s} (\vec{x}_i - ave(\vec{x})) \cdot (\vec{y}_{i+s} - ave(\vec{y})) & s < 0 \end{cases} \tag{3}$$

By calculating on different delay s, we can acquire a correlation score queue $cs_s(\vec{x}, \vec{y}) = (cs_1, ..., cs_m)$, where $n = 2m$, $s \in [1, m]$. The ultimate correlation score is called Key Correlation Distance (KCD). The $KCD(\vec{x}, \vec{y})$ is chosen from the multiple items of the cs queue in the following manner:

$$KCD(\vec{x}, \vec{y}) = max(\frac{cs_s(\vec{x}, \vec{y})}{\|\vec{x}_{n-s}\|_2 \cdot \|\vec{y}_{n-s}\|_2}) \tag{4}$$

In the cloud-database scenario, a low $KCD(\vec{x}, \vec{y})$ implies a high likelihood of abnormal KPI trends. After resolving the issue of computing the correlation scores of two time series with point-in-time delays, we need to find a way to preserve the correlation relationship among multiple databases in a unit on multiple KPIs. This correlation relationship is maintained using multiple Correlation Matrixes (CM), which are denoted as:

$$CM_j = \begin{pmatrix} 1 & \cdots & KCD(D_1, D_i) \\ & 1 & \cdots \\ & & 1 \end{pmatrix} \tag{5}$$

where $i \in [1, N]$. N indicates the maximum number of databases (databases can be flexibly expanded). D_i denotes the i-th database in a unit, and each item in CM_j is the KCD obtained by certain two databases on the j-th KPI in a given time window.

If the number of KPIs is Q, then there are Q correlation matrices. If there exists an unused database under a low workload scenario, all of its KPIs' correlation scores are set to 0. Since the correlation matrix is a symmetric matrix, there is no need to save the information of the lower triangular matrix. Multiple correlation matrices preserve the correlation relationship on multiple KPI time series in a unit.

C. Flexible Time Window Observation

In this section, we will describe how to observe a database state at different moments. We adopt a flexible time window observation mechanism that dynamically adjusts the window size to mitigate the effect of temporal fluctuations on accurately obtaining correlation scores.

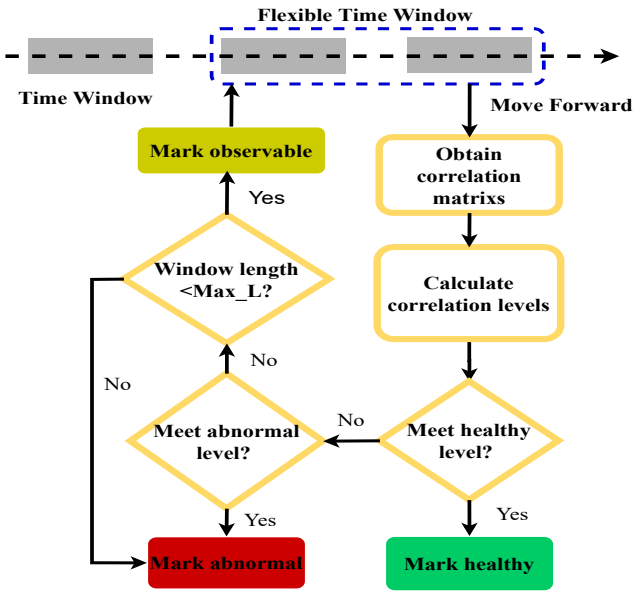


Fig. 7. Flow chart of database state determination. When a database is marked as "observable", the time window is automatically expanded. When the database is marked as a "healthy" state or "abnormal" state, this round of judgment process ends.

Algorithm 1: Get correlation levels

Input: α:correlation thresholds
θ:tolerance threshold
M:correlation matrixes
Output: correlation level dictionary D

```
D ← empty dictionary, Q ← number of KPIs
N ← number of databases in a unit, j ← 0, i ← 0
while i < Q do
    while j < N do
        D[j,i] ← CaculateLevels(M[i], α_i, θ, j)
        j ← j+1
    end
    i ← i+1
end
return D

CaculateLevels (matrix m, correlation threshold α_i,
  tolerance threshold θ, database number j):
KCDS = Search(m, j), k ← 0, L ← empty list
while k < N-1 do
    L[k] ← ScoreToLevel(KCDS[i], α_i, θ)
    k ← k+1
end
return L
```

We defined a correlation level to serve as a transition to determine the database state. Algorithm 1 shows the process of determining the correlation level. If a database exists but not in use (database expanded or reduced dynamically), it doesn't participate in the calculation of the correlation level. We use the following three steps to identify a database state:

Step1 Collecting correlation matrices. We calculate `Q` correlation matrices M through Equation (5), and `M[i]` represents the correlation matrix (CM) composed of the i-th KPI. After that, `M[i]`, the correlation thresholds α, a tolerance threshold θ and database number j are put into the function `CaculateLevels` to get the correlation level information.

Step2 Obtaining correlation level. In `CaculateLevels`, the correlation score list `KCDS` of the j-th database about the i-th KPI is first obtained via the function `Search`. Then the correlation scores in `KCDS` are converted into correlation levels via the function `ScoreToLevel`. We classify the KPIs' time series correlation scores into three levels: `level-1` means extreme deviation, `level-2` means slight deviation, and `level-3` means correlation. If the correlation score is less than the threshold α_i, the correlation score is converted to `level-1`. If the correlation score is between α_i and $\alpha_i - \theta$, the correlation score is converted to `level-2`. If the correlation score is greater than α_i, the correlation score is converted to `level-3`. Finally, `CaculateLevels` returns a correlation level, which are stored in the dictionary `D`.

Step3 Determining database state. The state of the database in that time window is determined by the number of correlation levels in D. In Figure 7, we describe how to determine a database state. If the number of `level-1` is greater than 0, the database state in the time window is considered "abnormal". If the database has KPIs at `level-2` and the number is fewer than a maximum tolerance deviation number, the database state for that time window is marked as "observable". If all KPIs are at `level-3`, the database state in that time window is marked as "healthy".

It's worth noting that the ultimate state is just "healthy" or "abnormal", the "observable" is only served as a transitional state. If a database is in an "observable" state, the streaming detection module will expand the time window. The size of the initial time window `W` is initially set according to the requirement for real-time, and generally ranges from 15 to 25 points. The length Δ of each expansion of the time window is generally the same as the initial window size. When the database state is "observable", `W` is updated to `W`+Δ. DBCatcher waits for data points to calculate the database state again. This process is repeated until the database state changes, or `W` exceeds the maximum window size W_M.

We calculate the average number of data points in all time windows and find that only a small number of time windows are scaled up to at most 2-3 times their initial size. The average time window remains almost unchanged, which has minimal impact on detection efficiency.

D. Adaptive Threshold Learning

The flexible window observation mechanism relies on multiple thresholds in the judgment process. Typically, thresholds need to be selected when creating databases or workload drifts. In this section, we will answer how to determine appropriate thresholds for multiple KPIs.

Algorithm 2: Adaptive threshold learning

Input: θ_{old}:genne thresholds
N:number of iterations
M:number of individuals
Output: new correlation thresholds θ_{new}

```
n ← 0, θ_best ← 0
max_performance ← θ_old_performance
while n < N do
    Get Individuals Performance;
    if θ_best > max_performance then
        Save θ_best
        max_performance ← θ_best_performance
    end
    Evict Poor Performance Individuals;
    Execute Selection Strategy;
    Execute Crossover Strategy;
    Ececute Mutation Strategy;
end
return θ_best
```

Algorithm 2 describes our threshold selection method in detail, which uses the genetic algorithm [31]. The genetic algorithm initializes multiple individuals with different genes. An individual's gene consists of three components: multiple correlation thresholds α_i, a tolerance threshold θ, and a maximum tolerance deviation number N. Individuals' genes determine the detection performance. Initially, multiple individual genes are generated randomly within a certain range. Afterward, all individuals calculate their detection performance based on the most recent period of judgment records. Then, retaining the genes of individuals with historically optimal detection performance. After that, the algorithm evicts the individuals with poor detection performance. Then, executing a crossover strategy and a mutation strategy among individuals with certain selection strategies to generate multiple new individuals, keeping the overall number of individuals constant. Following that, we will describe in detail how to implement the selection strategy, the crossover strategy, and the mutation strategy, which are key parts of genetic algorithms.

Selection Strategy. The probability of selecting an individual is determined by the individual's detection performance. We choose one individual called K_i, and the selection probability of K_i is denoted as:

$$P_i = \frac{D(K_i)}{\sum_{i=1}^{n} D(K_i)} \tag{6}$$

where D is the detection performance obtained by K_i. With a higher detection performance, K_i will have a larger chance of being chosen.

Crossover Strategy. The number of correlation thresholds for each individual is N. We randomly select M non-repeating numbers to construct a list $a = \{1, 2..., M\}$, where $M \in (0, N)$. For two selected individuals x and y, their correlation thresholds are denoted as: $x_c = \{\alpha_1^x, \alpha_2^x, ..., \alpha_N^x\}$, $y_c = \{\alpha_1^y, \alpha_2^y, ..., \alpha_N^y\}$. After crossover, two new individuals x^n and y^n are generated, and the correlation thresholds of x^n and y^n are denoted as : $x_c^n = \{\alpha_1^x, ...\alpha_m^x, \alpha_{m+1}^y, ..., \alpha_N^y\}$, $y_c^n = \{\alpha_1^y, ..., \alpha_m^y, \alpha_{m+1}^x, ..., \alpha_N^x\}$. Besides correlation thresholds, the tolerance threshold θ and the maximum tolerance deviation number N of new individuals x^n and y^n are chosen randomly from x and y.

Mutation Strategy. In order to explore the remaining threshold space extensively, the genetic algorithm mutates multiple individuals by a certain probability β. The correlation thresholds of new individuals are mutated at the correlation thresholds of the original individuals. Suppose a certain original correlation threshold is α_i and the new threshold randomly increases or decreases with a learning rate Δ. Besides correlation thresholds, the tolerance threshold θ and the maximum tolerance deviation number N of new individuals x^n and y^n are generated randomly within the respective range.

In general, each initial parameter is set in a certain range, where each initial correlation threshold $\alpha_i \in [0.6, 0.8]$, the tolerance threshold $\theta \in [0.1, 0.3]$, tolerance deviation number $N \in [0, 3]$, initial window size $W \in [15, 25]$, maximum window size $W_M \in [45, 75]$, and learing rate $\Delta = 0.1$. With the help of genetic algorithms, more suitable thresholds are continuously obtained based on recent judgement records.

IV. Experimental Evaluation

In this section, we will demonstrate that DBCatcher addresses the following three research questions:
RQ1: How well does DBCatcher perform in cloud database multivariate time series for anomaly detection? (§IV-B)
RQ2: Does DBCatcher get excellent detection performance and efficiency under different workloads? (§IV-C)
RQ3: What are the advantages of these three key technologies over existing methods in terms of detection performance and detection efficiency? (§IV-D)

A. Experimental design

*1) **Mixed Datasets**:* We use the following three datasets to verify the efficiency of DBCatcher: Tencent dataset, KPI time series come from multiple databases supporting various business scenarios, including but not limited to social networks, e-commerce, and finance; Sysbench and TPCC datasets, KPI time series obtained from verifiable open source test benchmarks to demonstrate the verifiability of the system. We proportionally inject the Sysbench and TPCC datasets with the time series deviations induced by the real Tencent cloud database abnormal issues. We give the application programming interface and visualization interface for obtaining KPI time series on Tencent Cloud [32], [33]. Table III shows the number of database units gathered, the number of KPIs, the number of time points, and the percentage of abnormal time points for each dataset. KPI information is presented in Table II (§II-B).

*2) **Irregular and Periodic Datasets**:* We classify these three datasets internally into irregular and periodic datasets to demonstrate the adaptability of different approaches under

TABLE III
STATISTICAL INFORMATION OF DIFFERENT DATASETS.

Dataset	Tencent	Sysbench	TPCC
No. of Units	100	50	50
No. of Dimensions	14	14	14
Total Points	5529600	648000	648000
Anomal Points	171970	27280	26308
Abnormal Ratio	3.11%	4.21%	4.06%

TABLE IV
TEST PARAMETER SPACE FOR SYSBENCH AND TPCC.

Dataset	Table	Thread	Item	Time(m)
Sysbench I	5-20	4-64	100000	0.5-1
Sysbench II	10	4-8-16-32	100000	0.5
Dataset	Warehouse	Thread	Warmup(m)	Time(m)
TPCC I	5-20	4-24	0.5-1	0.5-1
TPCC II	10	4-8-16-24	0.5	0.5

various workloads. We use the RobustPeriod [34] method to classify the Tencent dataset into irregular and periodic datasets based a KPI "Requests Per Second". Construct irregular datasets Sysbench I and TPCC I using the test space of table Table IV. Similarly, the periodic datasets Sysbench II and TPCC II datasets are constructed using the test space of Table IV. Periodic datasets account for 40% of the Tencent dataset, while irregular datasets account for 60% of the total. Therefore, we also generate the Sysbench and TPCC datasets according to the same irregular and periodic type proportions.

*3) **Evaluation Metrics**:* "Observable" is only an intermediate state, we use the ultimate states "healthy" and "abnormal" as the result of each time window. We use True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) to label an anomaly detection result. We evaluate performance and efficiency using the four generally used metrics listed below.

Precision. Precision is calculated as $\frac{TP}{TP+FP}$, which represents the proportion of correctly detected "abnormal" samples to all samples judged as "abnormal" by the method. The higher the Precision is, the better the method is.

Recall. Recall is calculated as $\frac{TP}{TP+FN}$, which represents the proportion of correctly detected "abnormal" samples to those marked as "abnormal" by database administrators. The higher the Recall is, the better the method is.

F-Measure. F-Measure is calculated as $\frac{2*Precision*Recal}{Precision+Recall}$. F-Measure is a balanced metric between Precision and Recall. The higher the F-Measure is, the better the method is.

Window-Size. The Window-Size indicates the size of the data points required for the time window in single detection. When the size of data points doesn't meet the Window-Size, the detection task is blocked. Therefore, Window-Size determines the speed of detecting anomalies and represents the detection efficiency. The smaller the Window-Size is, the higher the method detection efficiency is.

*4) **Compared Methods**:* We have selected some statistical and machine-learning based time series anomaly detection methods to compare with DBCatcher. We will give a brief introduction to these baseline methods.

Fast Fourier Transform (FFT) [7]. To acquire significant changes in single time series, FFT decomposes the single time series into separate components at several frequencies and then measures the degree of difference between time series points and surrounding points.

Spectral Residual (SR) [8]. Database anomalies can cause deviations in the trend of time series changes, which is the most intuitive part of the time series image. SR can detect these salient parts in single time series to identify issues.

SR-CNN [14]. SR-CNN is the base method of Microsoft anomaly detection service. Microsoft combines the SR method and convolutional neural networks (CNN) to further amplify the abnormal features of single time series.

OmniAnomaly [15]. OmniAnomaly combines GRU [35] (a variant of Recurrent Neural Network) and Variational Auto-Encoder [36] to model the temporal dependence and stochasticity of multivariate time series.

JumpStarter [16]. JumpStarter adopts a compressed sensing method [37] to quickly learn multivariate time series variation characteristics and an outlier-resistant sampling algorithm to reduce misclassification.

*5) **Database Environment**:* Tencent dataset comes from databases that support different online applications. For the Sysbench dataset and TPCC dataset, we apply for multiple units from Tencent Cloud [25]. Each unit contains one primary database and four replica databases. The database version number is MYSQL 5.7 and each database is configured with 4-cores, 8GB RAM, and 50GB disks.

*6) **Implementation**:* DBCatcher is implemented using Python 3.6 and DBCatcher is conducted on Tencent's cloud server with 12-core 4.0GHz CPU, 64GB RAM, and 200GB Disk.

B. Performance & Efficiency

We divide each mixed dataset into a training set and a testing set to evaluate the performance and efficiency of DBCatcher and other methods. In each mixed dataset, the first 50% of the time series are used as the training set and the remaining 50% of the time series are used as the testing set. For univariate time series anomaly detection methods, FTT, SR, SR-CNN, we first concatenate the same KPI time series in a unit into a one-dimensional time series. After that, to apply these methods to multivariate time series anomaly detection, we define the following simple rules: for an M-dimensional time series, in the time window t, if at least k ($1 \leq k \leq M$, where k can be tuned by different methods) univariate time series is "abnormal", then the time window is declared as "abnormal". For multivariate time series anomaly detection methods, OmniAnomaly, JumpStarter, we concatenate the same KPI from different databases into a one-dimensional time series, such that multiple one-dimensional time series constitute a multivariate time series. Each method uses the training set to randomly search thresholds and Window-size for which the optimal F-Measure can be obtained, and maintain them for evaluation on the testing set. Cloud database abnormal issues can result in financial loss and reputational impact for

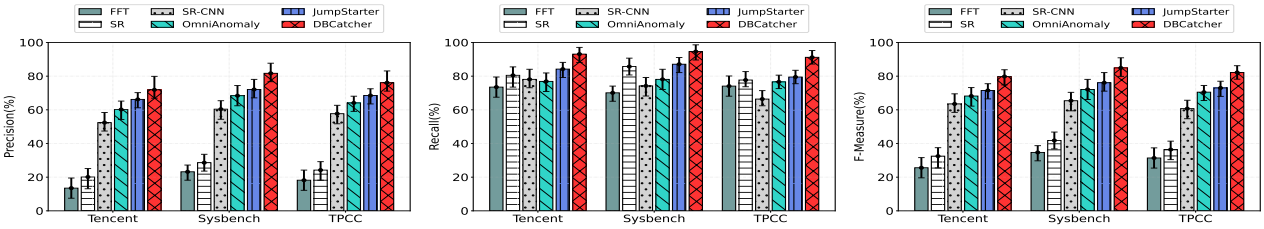


Fig. 8. Performance comparison of DBCatcher and other baseline methods on testing sets of three mixed datasets. We randomly search for optimal results in the training sets and maintain these thresholds in the testing sets. We perform 20 times and obtain the mean, maximum, and minimum values of the performance. DBCatcher obtains the best Precision, Recall, and F-Measure.

TABLE V
AVERAGE WINDOW-SIZES FOR DIFFERENT METHODS TO OBTAIN THE BEST F-MEASURE ON MIXED DATASETS.

	Tencent	Sysbench	TPCC
Model	Size	Size	Size
FFT	90	70	70
SR	70	60	50
SR-CNN	40	50	55
OmniAnomaly	70	60	50
JumpStarter	60	50	50
DBCatcher	20	20	20

TABLE VI
TRAINING TIME OF DIFFERENT ANOMALY DETECTION METHODS ON MIXED DATASETS.

	Tencent	Sysbench	TPCC
Model	Time(s)	Time(s)	Time(s)
FFT	525	354	454
SR	656	384	589
SR-CNN	4589	2462	2865
OmniAnomaly	3423	2106	2523
JumpStarter	2423	1523	1656
DBCatcher	1106	731	863

the cloud service vendors. Therefore, we generally consider detection performance to be more important than detection efficiency. We use the Window-Size when obtaining the best F-Measure to represent the detection efficiency.

*1) **Performance**:* Figure 8 shows the Precision, Recall and F-Measure obtained by DBCatcher and other baseline methods on three mixed datasets. FFT and SR are better suitable to detecting abnormal issues where the time series has salient variations, but in cloud-database scenario such time series may not represent abnormal issues, which can lead to extensive misclassifications. Thus, FFT and SR achieve a high Recall, but at the cost of low Precision. SR-CNN further improves performance, but is still limited to anomaly detection in univariate time series and can't discover the implicit association among multiple KPIs. OmniAnomaly is able to discover implicit association among multiple KPI, but OmniAnomaly has two limitations that hinder it from getting better performance: Firstly, OmniAnomaly requires extensive accumulation of data points for initialization; Secondly, OmniAnomaly requires multivariate time series vary minimally and with a certain degree of regularity. JumpStarter is able to solve the first limitation by a compressed sensing method [37], and also reduces misclassification by an outlier-resistant sampling algorithm [16]. However, JumpStarter still struggles to achieve significant performance on frequently changing workloads and time series with delays. DBCatcher is concerned with the correlations, which is less affected by changes in workload.

*2) **Efficiency**:* Table V shows the average window size for different methods to obtain the best F-Measure on mixed datasets. FFT and SR have to use a large Window-size in order to reduce misclassification, so they are tough to achieve high efficiency. SR-CNN improves the detection efficiency by amplifying the abnormal features in time series images to discover the abnormal time series trend. OmniAnomaly and JumpStarter need to learn normal variation patterns of time series, where small Window-size affects the learning effect and large Window-size may lead to overfitting. Although DBCatcher adopts a flexible time window observation method, it only expands a small number of time windows and the average Window-size remains almost unchanged. Table VI shows the training time of different methods on the three datasets. Although the training time of DBCatcher is higher than that of FFT and SR, the F-measure obtained by DBCatcher is optimal. SR-CNN, OmniAnomaly and JumpStarter can also obtain decent F-Measure, but the training time is also relatively longer.

Compared with the optimal JumpStarter, DBCatcher improves F-Measure by 8.3%, 8.8%, and 9.2% on the three datasets. Meanwhile, DBCatcher maintains a high online detection efficiency and an acceptable training time.

C. Workload Adaptability

We first apply different methods on irregular and periodic datasets to verify their adaptability under different workloads. Then, we evaluate the retraining time required by different methods when the workload drifts.

*1) **Irregular datasets**:* We first compare DBCatcher with baseline methods on different irregular datasets. We conduct experiments and show the performance results in Figure 9. The experimental results show a decrease in F-Measure obtained by most methods except for DBCatcher. FFT, SR, and SR-CNN are tougher to accurately identify abnormal issues on complex irregular datasets than on mixed datasets. OmniAnomaly and

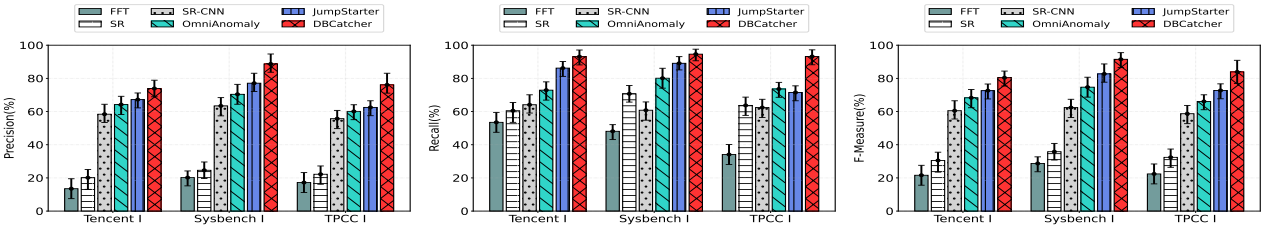


Fig. 9. Performance comparison of DBCatcher and other baseline methods on testing sets of three irregular datasets. We perform 20 times and obtain the mean, maximum, and minimum values of the performance. DBCatcher obtains the best Precision, Recall, and F-Measure.

TABLE VII
AVERAGE WINDOW-SIZES FOR DIFFERENT METHODS TO OBTAIN THE BEST F-MEASURE ON IRREGULAR DATASETS.

	Tencent I	Sysbench I	TPCC I
Model	Size	Size	Size
FFT	100	70	80
SR	80	60	70
SR-CNN	60	70	60
OmniAnomaly	80	60	60
JumpStarter	60	50	60
DBCatcher	20	20	20

TABLE VIII
WINDOW-SIZES FOR DIFFERENT METHODS TO OBTAIN THE BEST F-MEASURE ON PERIODIC DATASETS.

	Tencent II	Sysbench II	TPCC II
Model	Size	Size	Size
FFT	70	50	45
SR	50	40	50
SR-CNN	50	40	40
OmniAnomaly	65	50	50
JumpStarter	60	50	50
DBCatcher	20	15	20

JumpStarter focus on learning normal variation patterns in time series, but this pattern variation may not be apparent on irregular datasets. Compared to OmniAnomaly, JumpStarter adopts an outlier-resistant sampling algorithm [16] that enhances the robustness of anomaly detection to a certain extent. The advantage of DBCatcher is that it focuses on the correlations of time-series changes in KPIs across databases in a unit and is insensitive to changes in workloads. We record the Window-size for different methods to obtain the best F-Measure, as shown in Table VII. We find that most methods on irregular datasets may require a longer time window compared to that on mixed datasets. Irregular variation patterns result in baseline methods having to improve detection performance by reducing detection efficiency.

2) ***Periodic datasets****:* We then proceed to evaluate DBCatcher and other baseline methods on periodic datasets. On Tencent II dataset, Sysbench II dataset, and TPCC II dataset, we calculate the Precision, Recall, and F-Measure of these methods displayed in Figure 10, respectively. We find that SR and SR-CNN have a large performance improvement on periodic datasets, which is due to the fact that abnormal features of periodic time series are more easily identified.

TABLE IX
DBCATCHER RETRAINING TIME WHEN WORKLOAD DRIFTS.

	T-S	T-C	S-C
Model	Time(s)	Time(s)	Time(s)
FFT	318	212	298
SR	456	216	315
SR-CNN	3658	2151	2591
OmniAnomaly	2848	1698	2425
JumpStarter	1855	1289	1513
DBCatcher	625	459	593

Note: T-S represents the drift from Tencent to Sysbench. T-C represents the drift from Tencent to TPCC. S-C represents the drift from Sysbench to TPCC.

In addition, OmniAnomaly and JumpStarter are also better at anomaly detection on periodic datasets compared to irregular datasets. We record the Window-size for different methods to obtain the best F-Measure, as shown in Table VIII. Since the time series changes more regularly, the Window-sizes of FFT and SR are significantly reduced. While SR-CNN, OmniAnomaly, and JumpStarter need a large number of data points to learn variation patterns, so the reductions in Window-size is not obvious. DBCatcher can still achieve significant detection performance and detection efficiency on periodic datasets, because if KPI time series don't have periodic variation patterns, KPI time series trend will deviate from the normal time series trend.

3) ***Workload drifts****:* Cloud database workloads are user-determined and can be changed at any time. Therefore, the workload adaptability of anomaly detection methods is more critical for cloud databases than for other online service systems. To do this, we test the retraining time of different methods when the workload shifts (shown in Table IX). According to Table IX, compared with machine-learning methods, DBCatcher requires far less retraining time when the workload drifts, which indicates that DBCatcher can adjust the threshold faster and better adapt to different workloads.

D. Key Technologies Affects

We have adopted three core technologies to improve the detection performance of DBCatcher. This section compares the performance of these techniques with existing techniques for multivariate time series anomaly detection in cloud databases.

1) Efficient time series correlation measurement: We choose the KCD as the method for comparing two time series. We evaluate KCD with pearson coefficients [29], dynamic time

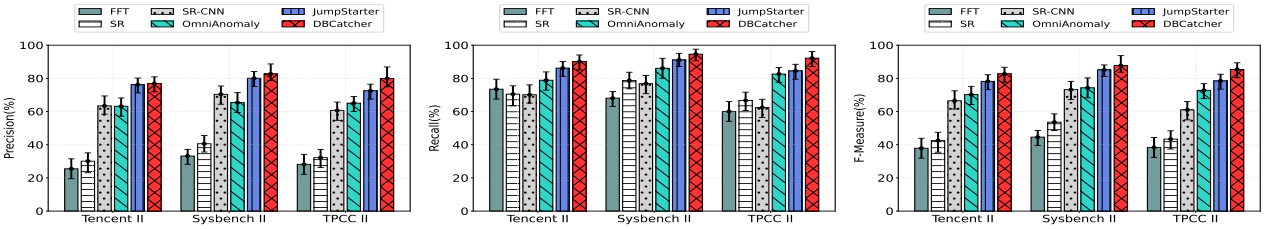
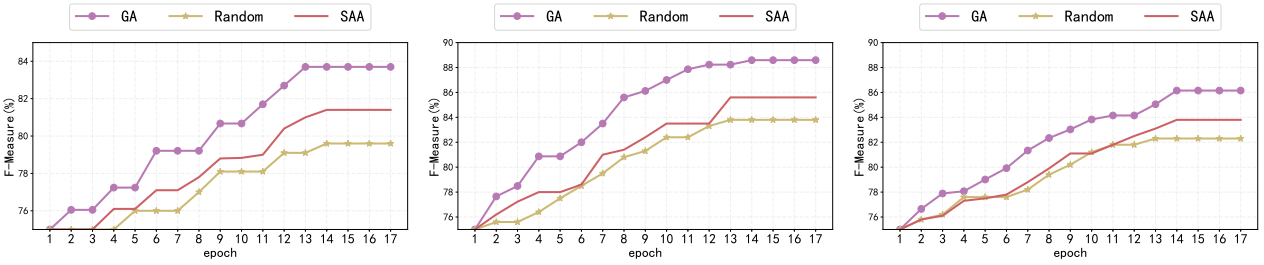


Fig. 10. Performance comparison of DBCatcher and other baseline methods on testing sets of three periodic datasets. We perform 20 times and obtain the mean, maximum, and minimum values of the performance. DBCatcher obtains the best Precision, Recall, and F-Measure.



(a) Tencent mixed dataset (b) Sysbench mixed dataset (c) TPCC mixed dataset

Fig. 11. Performance of genetic algorithm (GA), simulated annealing algorithm (SAA), and random search algorithm (Random). Each method is tested 20 times on different datasets to obtain the average value of F-Measure. GA achieves the best F-Measure.

TABLE X
F-MEASURE VALUES FOR DIFFERENT CORRELATION MEASUREMENT METHODS COMBINED WITH MM.

	Tencent	Sysbench	TPCC
Model	F-Measure(%)	F-Measure(%)	F-Measure(%)
MM-Pearson	69.2	72.4	67.1
MM-DTW	58.1	67.3	61.2
MM-KCD	74.5	76.8	77.7
AMM-KCD	79.5	83.9	82.1

Note: AMM-KCD indicates the addition of a flexible time window observation method to MM-KCD.

warping (DTW) [30] on three mixed datasets, showing the detection performance in Table X. Pearson coefficient doesn't take into account the delay among data points, resulting in a decrease in F-Measure. Dynamic time warping looks for the most "similar" point in two time series, but this can result in an inconsistent number of delays at each point in a time window. This is the opposite of the cloud-database scenario, where data point delays should be essentially the same in a time window.

2) Flexible time window observation: In the cloud-database scenario, temporal fluctuations in KPI time series are unavoidable. Therefore, sometimes we need a longer time window to determine a database state. Table X shows the performance enhancement of the flexible time window observation mechanism on different mixed datasets. While the flexible time window mechanism may reduce detection efficiency, it also enhances detection performance, depending on which metric users are more concerned about.

3) Adaptive threshold learning: The adaptive threshold learning policy can assist DBCatcher to find suitable thresholds. We compare the genetic algorithm with the random search method and the simulated annealing method and display the results in Figure 11. In general, we set the minimum F-Measure criterion to 75% (can be set according to different situations). The adaptive threshold learning policy will only be activated if the original thresholds don't meet this criterion.

4) Component computation time: We apply DBCatcher to 50 units, each containing five databases. We record the average computation time per component when DBCatcher performs online detection tasks. For a 100M dataset, corresponding to the amount of data for 120 hours of KPI data points, DBcatcher takes only 42 seconds to complete all detection tasks. On average, the efficient time series correlation measurement method accounts for 70% of the overall time and the flexible time window observation method accounts for 30%. The adaptive threshold learning method is performed only when the database is created or when the workload drifts.

V. CASE STUDY AND DISCUSSION

Case Study. To demonstrate the practicality of DBCatcher on cloud databases, we list two real anomaly cases found by DBCatcher on the Tencent Cloud Database. These two anomaly cases are tough to detect with existing methods and to be noticed by DBAs.

Figure 12 shows cloud databases with a `level-1` anomaly. DBAs check the table information of databases and find that there is a huge gap between the amount of data stored and the actual capacity occupied. The reason is that these databases have executed a lot of delete and insert operations. Due to the negligence of DBAs, the storage space is severely fragmented. `Level-1` anomalies mainly occur in critical KPIs such as

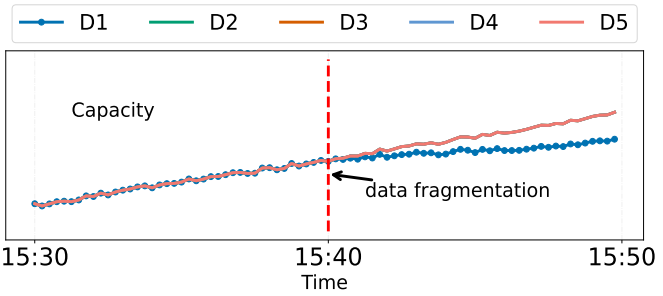


Fig. 12. At 15:40, there is a different trend in "capacity" within the unit. The reason is that other databases generate a lot of data fragments.

reads, writes, and capacity. Any problem with these KPIs can cause a serious accident in the database.

Figure 13 illustrates a `level-2` anomaly that occurs in an e-commerce transaction scenario. DBAs observe that the `Total Requests` are basically the same between databases, but the `CPU Utilization` for `D1` increases twice as much as for the other databases. The `Innodb Rows Read` of `D1` is also significantly different from other databases. The reason is that although different databases have the same number of requests, the resources consumed per request can vary greatly. It happens that at this time a large number of resource-consuming tasks are mapped to `D1`. There are numerous reasons for `level-2` anomalies, such as hardware problems, network lag, and slow queries.

Strengths and weaknesses. The strengths of DBCatcher over existing methods are the ability to implement online anomaly detection when DBMS meets UKPIC phenomena, and the ability to quickly adapt to workload changes. However, DBCatcher appears to be powerless for multiple databases with simultaneous anomalies (although such cases are rare). Also, DBCatcher will not work if the KPIs affected by the anomaly do not break the UKPIC phenomenon. Therefore, DBCatcher complements existing methods for more comprehensive detection of cloud database anomalies.

Future work. We will continue our research on the following two aspects: (1) What are the common anomalies in cloud databases? How can we combine existing anomaly detection methods to provide better anomaly detection services? (2) After detecting anomalies, how can root cause analysis be performed using database KPI time series?

VI. Related Work

Anomaly detection. Existing approaches include: (1) Statistical based method: FFT [7], SR [8], Wavelet Transform [38], RRCF [39], and EGADS [5]. The detection performance obtained by these methods is hardly adequate for practical industrial applications. (2) Machine-learning based method: Opprentice [40], SR-CNN [14], DONUT [18], OmniAnomaly [15], and JumpStarter [16]. These methods learn the variation patterns of time series through a large amount of data and lose detection efficiency. Different from them, DBCatcher focuses on time series correlation and can detect anomalies more accurately and efficiently.

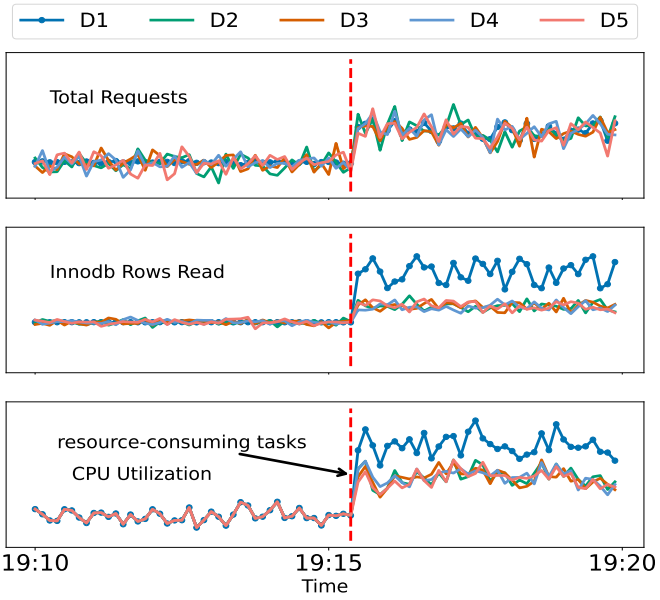


Fig. 13. At 19:16, a database within the unit performs a large number of resource-consuming tasks.

Correlation measurement. Pearson coefficient [29] is used to evaluate the linear relationship between two time series but doesn't consider data point delays. Dynamic time warping [30] causes mismatching of time points, resulting in inaccurate calculation of correlation scores. Spearman coefficient [41] only focuses on whether there is a simple monotonic relationship between two time series. CoFlux [42] calculates correlations based on the ability to make accurate forecasts of time series, requiring a stable pattern of change in time series.

VII. Conclusion

Numerous studies have demonstrated that abnormal time series trends in key performance indicators are often a precursor to cloud database problems. DBAs can't observe the key performance indicator time series trends one by one, since this is very labor-intensive and time-consuming. In this paper, we discover a phenomenon named unit key performance indicator correlation in cloud database architecture. Based on this phenomenon, we propose an efficient cloud database online anomaly detection system called DBCatcher, which can accurately discover cloud database issues. We use real-world and synthetic datasets to demonstrate that DBCatcher offers a better balance among detection performance, detection efficiency, and workload adaptability than existing methods.

Acknowledgement

We thank our shepherd and the anonymous reviewers for their valuable comments and helpful suggestions. This work is supported in part by the National Natural Science Foundation of China (Grant No.62232007, No.61821003). Chunhua Li is the corresponding author of the paper.

REFERENCES

[1] Gartner., "Cloud database market," 2021. [Online]. Available: https://www.datamation.com/cloud/cloud-database-market/

[2] F. Li, "Cloud-native database systems at alibaba: Opportunities and challenges," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2263–2272, 2019.

[3] 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*, ser. SIGMOD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1599–1614.

[4] 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," *Proc. VLDB Endow.*, vol. 13, no. 8, p. 1176–1189, apr 2020.

[5] N. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1939–1947.

[6] 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)*, 2018, pp. 13–24.

[7] C. Van Loan, *Computational frameworks for the fast Fourier transform*. SIAM, 1992.

[8] X. Hou and L. Zhang, "Saliency detection: A spectral residual approach," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[9] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 387–395.

[10] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1544–1551, 2018.

[11] Z. Li, Y. Zhao, J. Han, Y. Su, R. Jiao, X. Wen, and D. Pei, "Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 3220–3230.

[12] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 1409–1416.

[13] Y. Su, Y. Zhao, M. Sun, S. Zhang, X. Wen, Y. Zhang, X. Liu, X. Liu, J. Tang, W. Wu *et al.*, "Detecting outlier machine instances through gaussian mixture variational autoencoder with one dimensional cnn," *IEEE Transactions on Computers*, 2021.

[14] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at microsoft," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 3009–3017.

[15] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2828–2837.

[16] M. Ma, S. Zhang, J. Chen, J. Xu, H. Li, Y. Lin, X. Nie, B. Zhou, Y. Wang, and D. Pei, "{Jump-Starting} multivariate time series anomaly detection for online service systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 413–426.

[17] Twitter., "Twitter engineering: Introducing practical and robust anomaly detection in a time series." [Online]. Available: https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.html

[18] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, and H. Qiao, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, p. 187–196.

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

[20] Y. Zhang, C. Ruan, C. Li, X. Yang, W. Cao, F. Li, B. Wang, J. Fang, Y. Wang, J. Huo *et al.*, "Towards cost-effective and elastic cloud database deployment via memory disaggregation," *Proceedings of the VLDB Endowment*, vol. 14, no. 10, pp. 1900–1912, 2021.

[21] W. Cao, Y. Liu, Z. Cheng, N. Zheng, W. Li, W. Wu, L. Ouyang, P. Wang, Y. Wang, R. Kuan *et al.*, "{POLARDB} meets computational storage: Efficiently support analytical workloads in {Cloud-Native} relational database," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 29–41.

[22] Ł. Korycki and B. Krawczyk, "Concept drift detection from multi-class imbalanced data streams," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 1068–1079.

[23] amazon, "Amazon cloud database services." [Online]. Available: https://aws.amazon.com/cn/rds/features/read-replicas/

[24] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao, "Amazon aurora: Design considerations for high throughput cloud-native relational databases," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1041–1052.

[25] Tencent, "Tencentdb." [Online]. Available: https://cloud.tencent.com/product/cdb

[26] 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*, 2015, pp. 211–224.

[27] 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)*. IEEE, 2018, pp. 13–24.

[28] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–33, 2021.

[29] J. Cohen, "Statistical power analysis," *Current Directions in Psychological Science*, vol. 1, no. 3, pp. 98–101, 1992.

[30] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 262–270.

[31] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[32] Tencent, "Tencent cloud get key perfoemance indicator time series api." [Online]. Available: https://cloud.tencent.com/document/api/248/30351

[33] "Tencent cloud database intelligent monitoring service." [Online]. Available: https://cloud.tencent.com/product/dbbrain

[34] Q. Wen, K. He, L. Sun, Y. Zhang, M. Ke, and H. Xu, "Robustperiod: Robust time-frequency mining for multiple periodicity detection," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD/PODS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2328–2337.

[35] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling. arxiv 2014," *arXiv preprint arXiv:1412.3555*, 2014.

[36] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[37] Y. C. Eldar and G. Kutyniok, *Compressed sensing: theory and applications*. Cambridge university press, 2012.

[38] W. Lu and A. A. Ghorbani, "Network anomaly detection based on wavelet analysis," *EURASIP Journal on Advances in Signal Processing*, vol. 2009, pp. 1–16, 2008.

[39] S. Guha, N. Mishra, G. Roy, and O. Schrijvers, "Robust random cut forest based anomaly detection on streams," in *International conference on machine learning*. PMLR, 2016, pp. 2712–2721.

[40] 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*, ser. IMC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 211–224.

[41] S.-B. Lee, D. Pei, M. Hajiaghayi, I. Pefkianakis, S. Lu, H. Yan, Z. Ge, J. Yates, and M. Kosseifi, "Threshold compression for 3g scalable monitoring," in *2012 Proceedings IEEE INFOCOM*, 2012, pp. 1350–1358.

[42] Y. Su, Y. Zhao, W. Xia, R. Liu, J. Bu, J. Zhu, Y. Cao, H. Li, C. Niu, Y. Zhang, Z. Wang, and D. Pei, "Coflux: Robustly correlating kpis by fluctuations for service troubleshooting," in *Proceedings of the International Symposium on Quality of Service*, ser. IWQoS '19. New York, NY, USA: Association for Computing Machinery, 2019.