Identifying Root-Cause Metrics for Incident Diagnosis in Online Service Systems

Canhua Wu[†¶], Nengwen Zhao[†¶], Lixin Wang[‡], Xiaoqin Yang[‡], Shining Li[‡], Ming Zhang[‡], Xing Jin[‡], Xidao Wen[†¶], Xiaohui Nie[†¶], Wenchi Zhang[§], Kaixin Sui[§], Dan Pei[*†¶]
[†]Tsinghua University [‡]China Construction Bank [§]BizSeer
[¶]Beijing National Research Center for Information Science and Technology (BNRist)

***Abstract*—Incidents in online service systems could incur poor user experience and tremendous economic loss. To reduce the influence of incidents and guarantee service reliability, it is critical to identify root-cause metrics for engineers with clues to assist incident diagnosis. However, it is a challenging task due to the complicated dependencies and huge volume of various metrics in large-scale systems. Existing approaches are based on either anomaly detection or correlation analysis, performing not well in terms of accuracy or efficiency. To better understand the problem of root-cause metric identification, we conduct a preliminary study based on real-world data analysis and interactions with engineers. The key observation is that root-cause metrics should satisfy two requirements. One is that the metric is expected to behave abnormally during the incident; the other is that the anomaly pattern should meet physical meaning and engineers' demand. Motivated by the findings obtained from the study, we propose an effective approach named *PatternMatcher* to identifying root-cause metrics accurately. Specifically, *PatternMatcher* contains three steps, where coarse-grained anomaly detection aiming to filter out normal metrics, anomaly pattern classification aiming to filter out unimportant anomaly patterns, and root-cause metric ranking. An extensive study on four real-world datasets including 113 incident cases from a large commercial bank demonstrates that *PatternMatcher* outperforms all baseline approaches, achieving top-3 average accuracy of 0.91. Moreover, we have deployed *PatternMatcher* in practice and shared some successful cases from real deployment.**

***Index Terms*—Root-cause metric, incident diagnosis, anomaly pattern classification**

I. Introduction

Recently, online service systems have become increasingly popular in our daily life, such as search engines, online shopping, and social networks. To ensure service quality and user experience, various control measurements have been employed to guarantee that services are running normally [1], [2]. Nonetheless, incidents (or unplanned interruptions) are still inevitable in practice due to the complexity and the large scale of service systems [2]. These incidents might incur tremendous economic loss and user dissatisfaction. For example, a recent survey [3] reveals that the average cost of server downtime per hour is between $301,000 and $400,000. Therefore, it is critical to accurately locate the root cause after an incident occurs to restore the service as soon as possible.

In general, once an incident occurs, engineers tend to collect all related metrics and investigate their behaviors, aiming to find out the root cause or some clues for further diagnosis. For example, if the response time of a service increases dramatically, engineers would check the metrics of the infrastructure layer (e.g., network, server, and database) associated with the service. If we found that the CPU usage of a server is high, it indicates that the high latency may be incurred by server overload, and the incident could be mitigated by killing the high-load session or restarting the server. According to our observations, however, hundreds to thousands of metrics need to be inspected for one incident, and there could be tens of incidents one day in large-scale service systems. Thus, it is time-consuming and error-prone for engineers to manually check so many potentially relevant metrics to locate the root cause. Therefore, an automated root-cause metric identification approach is highly desired.

In the literature, many efforts have been devoted to automatically identifying root-cause metrics [4], [5], [6], [7], but they still suffer from various limitations in practice. First, the majority of the approaches use statistical anomaly detection methods to identify suspicious root-cause metrics such as KDE (Kernel Density Estimation) in FluxRank [4] and ϵ-statistic in ϵ-diagnosis [5]. However, these anomaly detection methods do not perform well in practice. Taking KDE for example, it is challenging for engineers to select appropriate kernel functions and bandwidths which are crucial for the overall accuracy. Besides, it is unreasonable to estimate a distribution given limited data points. Second, all approaches ignore the physical meaning of metrics and anomaly patterns. For example, a sudden decrease of CPU usage may be wrongly identified as a root-cause metric using existing works, while the CPU decrease pattern means more CPU resources are released, which is a good phenomenon and cannot incur incidents. Besides, different anomaly patterns may indicate different severity levels in diagnosis (§II-B), which should be considered in ranking root-cause metrics. Third, some approaches based on correlation analysis rely on long-term historical data [6], leading to efficiency and resource bottlenecks faced with thousands of candidate metrics in practice. Thus it is imperative to develop an effective and practical approach to identifying root-cause metrics.

To better understand the problem of identifying root-cause metrics in practice, we conducted a preliminary study based on large-scale real-world data and interviews with engineers. We obtain three key findings through the study. First, identifying root-cause metrics is indeed helpful for engineers to conduct

* Dan Pei is the corresponding author.

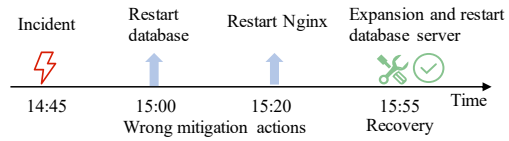
incident diagnosis, while the current practice in the real world is far from satisfactory (§II-B1). Second, anomaly patterns should be incorporated in analyzing root-cause metrics and can provide more comprehensive information for engineers to enhance the interpretability and performance by filtering out abnormal but unconcerned patterns (§II-B2). Third, diverse anomaly patterns exist in the real world, which may represent different severity levels and explanatory power to root cause, which should be considered in incident diagnosis (§II-B3).

Inspired by the insights learned from the study and to overcome the aforementioned limitations, we propose a novel root-cause metric identification approach named *PatternMatcher*. *PatternMatcher* contains three components: coarse-grained anomaly detection, anomaly pattern classification, and root-cause metric ranking. More specifically, *PatternMatcher* first conducts a two-sample test to filter out normal metrics to reduce search space. This is because root-cause metrics should behave abnormally when an incident occurs. After identifying anomalies, *PatternMatcher* draws support from 1-D CNN (One-Dimensional Convolutional Neural Network) [8] and classifies abnormal segments into different patterns, each of which indicates a different physical meaning. Besides, pattern classification could further filter out some false positive patterns that engineers are not concerned about to improve the accuracy. Finally, *PatternMatcher* sorts candidate metrics based on our designed ranking strategy, considering both anomaly scores in the first step and pattern weights in the second step. In this way, engineers could investigate the root cause according to our recommended ranking list.

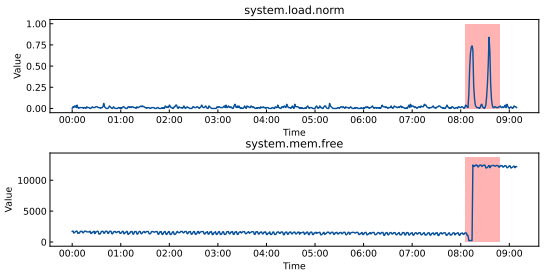
To illustrate the effectiveness of *PatternMatcher*, we conducted an extensive experimental study on four datasets containing diverse incidents (113 cases in total) from a large commercial bank which supports hundreds of millions of users and hosts hundreds of service systems. We compare *PatternMatcher* with several baseline approaches, i.e., FluxRank [4], ϵ-diagnosis [5], CETS [6] and MicroCause [7]. The $Avg@3$ accuracy of *PatternMatcher* could achieve 0.91 on average on four datasets. In comparison, four baseline approaches could only achieve 0.70, 0.44, 0.66 and 0.40. The contributions of two main components of *PatternMatcher* (anomaly pattern classification and ranking) are also confirmed. In terms of time efficiency, *PatternMatcher* could provide ranking results within dozens of seconds faced with thousands of candidate metrics, which is satisfactory for engineers in practice. More importantly, *PatternMatcher* has been deployed in a large commercial bank and engineers appreciate its value in assisting incident diagnosis. To show the practical usage of *PatternMatcher*, we introduce the deployment architecture and some successful cases collected from practice.

To summarize, this work has the following contributions:

- We conducted the first preliminary study to investigate the problem of identifying root-cause metrics and the necessity of considering anomaly patterns.
- We propose a novel approach named *PatternMatcher* to identifying root-cause metrics for incident diagnosis. The core component in *PatternMatcher* is an effective anomaly pattern classification based on 1-D CNN [8].
- Extensive experiments based on various real incidents demonstrate the effectiveness of *PatternMatcher*, achieving $Avg@3$ accuracy of 0.91 on four datasets on average and outperforming baseline approaches. Besides, *PatternMatcher* has been deployed in practice.



(a) An incident case with long TTM due to inaccurate root cause analysis



(b) A case to illustrate the importance of anomaly pattern

Fig. 1: Real-world case analysis

II. Motivation and Problem Formulation

A. Motivation Case

Identifying root-cause metrics for incident diagnosis is critical for online service systems, to guarantee short TTM (Time to Mitigate) and service reliability. To illustrate the importance of accurate root cause analysis, we present a real-world incident case in Fig. 1(a). This incident was noticed at 14:45 during busy hours. After preliminary diagnosis, engineers restarted the database and Nginx at 15:00 and 15:20, respectively, but the incident was not mitigated successfully. Finally, engineers located that the root cause was the capacity problem of the database server leading to system overload. The incident was solved at 15:55 by expanding the capacity and restarting the server. Overall, the total TTM is 70 minutes, during which the service quality and user experience were severely damaged. If engineers could notice the abnormal behaviors of related metrics (e.g., CPU usage) of the database server, the incident could be mitigated much earlier. This case vividly demonstrates that identifying root-cause metrics is valuable for incident diagnosis.

In addition to identifying the abnormal metrics, the specific anomaly pattern is also essential for incident diagnosis. Fig. 1(b) presents an intuitive example to illustrate the significance of anomaly patterns when analyzing the root cause. We observe that both two metrics (system.load.norm and system.mem.free) behave abnormally during the incident time (pink bands). However, after further investigating the anomaly patterns, we noticed that the system.mem.free could be excluded because the increase pattern of system.mem.free is often a positive phenomenon and is unlikely to be the root

cause. If engineers ignore it, they may take incorrect mitigation actions (e.g., expanding memory), resulting in a longer TTM.

Through actual case analysis, we find that identifying the root-cause metrics is vital for incident diagnosis to save TTM. In addition, we should pay extra attention to the specific abnormal behavior of different metrics to enhance the accuracy and interpretability of root cause analysis.

B. Preliminary Study

To better understand the problem of identifying root-cause metrics in practice, we conducted a preliminary study based on real-world data analysis and interviews with engineers. Specifically, we collected historical incidents and metrics of 14 online service systems from a large commercial bank. The total size of various metric data exceeds 50GB, and the total number of metrics is over 260K, including application (e.g., response time and success rate), storage (e.g., average write/read time), network (e.g., average round-trip time) and so on. To our best knowledge, this is the first study that focuses on root-cause metric identification and anomaly patterns. In this study, we aim to answer the following three questions:

- How about the current practice of identifying root-cause metrics?
- Are anomaly patterns necessary for identifying root-cause metrics?
- What types of anomaly patterns exist in practice?

1) Current Practice: We interviewed dozens of engineers from different teams in a large commercial bank. These engineers with rich experience frequently diagnose incidents in their daily work via inspecting various monitoring metrics. Thus the interview results are quite convincing. In current practice, we find that engineers tend to adopt some monitoring and visualization tools (e.g., Grafana [9] and Kibana [10]) to inspect the behavior of related metrics. In practice, the number of candidate metrics is way larger than that of what engineers can properly investigate. To address this issue, engineers draw support from some simple yet efficient anomaly detection methods (e.g., 3-σ) to filter out some normal metrics which are unlikely to have correlations with the incident. Nevertheless, it is still far from satisfactory due to the natural limitation of anomaly detection methods, and the number of abnormal metrics requiring manual examination is still large after the rough filtering process. Consequently, it is imperative to develop an automated, accurate, and efficient approach to identifying root-cause metrics for incident diagnosis.

2) The Necessity of Anomaly Pattern: According to the feedback of engineers, they confirmed the significance of anomaly patterns in the task of identifying root-cause metrics (and some other tasks in service reliability management). The reasons can be summarized as follows. First, providing specific anomaly patterns could assist engineers to obtain more comprehensive information. For example, an abrupt anomaly (e.g., spike) indicates a sudden incident, while a trending anomaly (e.g., steady increase) indicates a slowly deteriorating incident. Second, some anomaly patterns are not concerned by engineers, which could bring some inaccurate results and incurring long TTM. Existing time series anomaly detection works identify statistical anomalies but fail to consider the physical meaning of the metrics and corresponding anomalies. For example, the decrease in CPU usage and the increase in transaction volume are statistical anomalies, but they are positive phenomenons and cannot be root-cause metrics. Such abnormal but positive patterns should be further filtered to improve performance. As a result, anomaly patterns should be well incorporated when identifying root-cause metrics to enhance accuracy and interpretability.

3) Typical Anomaly Patterns: To answer this question, we analyzed numerous metrics from the real world. Through our analysis and discussions with engineers, we summarized 13 typical anomaly patterns, displayed in Fig. 2. We observe that anomaly patterns are indeed diverse in practice, and a metric could exhibit more than one anomaly pattern. Besides, we further analyzed the physical meanings of different anomaly patterns. As shown in Table I, we present detailed explanations of these anomaly patterns, including corresponding example metrics and problem scenarios. It is clear that different patterns may represent different problems. For example, the "steady increase" of memory usage may indicate a deteriorating capacity issue, and the "single spike" of response time may indicate that the system is jammed briefly but then recovered quickly. These patterns can be divided into two categories based on whether the metric resumes to a normal state in the end (Type-1 and Type-2). Interestingly, we observe that different anomaly patterns may represent different severity levels and explanatory power to root cause. Intuitively, Type-1 patterns are much more severe than Type-2 ones because the former means that the anomaly is still continuing, while the latter is a transient anomaly and recovers quickly.

4) Summary: Through this preliminary study, we obtain the following key findings that largely motivate our work.

- Identifying root-cause metrics is critical for troubleshooting, while the current practice in the real world is unsatisfactory.
- Incorporating anomaly patterns is indeed helpful for identifying root-cause metrics, to further enhance the accuracy and interpretability.
- There exist diverse anomaly patterns with different physical meanings and different explanatory power to root causes, which should also be noticed.

C. Problem Formulation

In this paper, we aim to propose an effective and efficient approach to identifying root-cause metrics to assist incident diagnosis. Specifically, when an incident (e.g., success rate decreases dramatically) occurs, all related metrics are collected, such as the infrastructure components supporting the service, including database, server, hardware and so on. Then, we expect to rapidly identify root-cause metrics from numerous candidate metrics and provide a ranking list to engineers, to guide further troubleshooting. Importantly, the root-cause metrics need to meet two requirements. One is that the metrics should behave abnormally during the occurrence

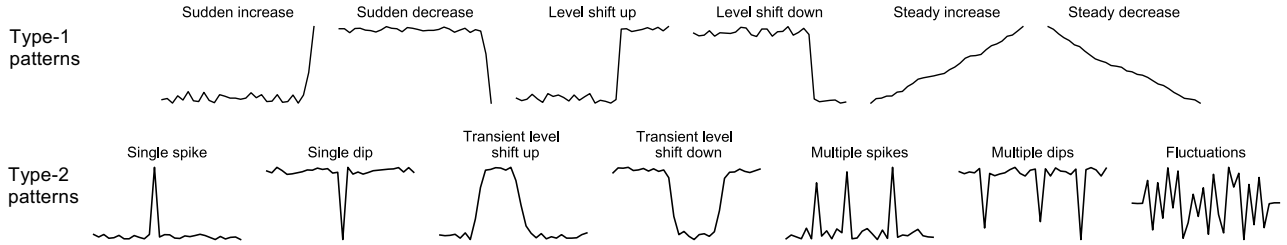


Fig. 2: Typical anomaly patterns summarized from large-scale real-world data

TABLE I: Detailed explanations of anomaly patterns, including corresponding example metrics and problem scenarios

Type	Anomaly patterns	Example metrics	Examples of problem scenarios
Type-1: Still in abnormal state	Sudden increase	system.cpu.pct_usage; system.mem.buffered; system.net.packets_in.error	A sudden high-load task occupies many resources; Cyclic calls due to configuration errors or bugs
	Sudden decrease	system.net.packets_in.count; weblogic.webapp.sessions; app.response.rate	Network disconnection; Request obstruction; Power outage
	Level shift up	system.mem.pct_usage; system.io.w_await	A sudden high-load task lasts for a while; Bursted user request
	Level shift down	system.cpu.idle; system.mem.free; app.success.rate	Increase in bad blocks on disk; Power outage; Server down
	Steady increase	system.disk.used; oracle.tablespace.used	Memory leak caused by buggy code; Insufficient resources
	Steady decrease	system.mem.free; oracle.tablespace.free	
Type-2: Recover to normal state	Single spike	oracle.lock.wait; system.disk.await	Transient network jitter; A temporary high-load task
	Single dip	system.net.bytes_rcvd; app.success.rate	Network/system stuck and recovery
	Transient level shift up	oracle.read.sequential; oracle.lock.wait	A high-load task takes up resources for a while and has been released; A faulty disk has been replaced
	Transient level shift down	system.cpu.idle;system.mem.free	
	Multiple spikes	system.io.await; app.response.time	Continuously unstable system; Network fluctuations
	Multiple dips	system.net.bytes_sent; app.success.rate	
	Fluctuations	system.mem.used;app.response.rate; system.cpu.pct_usage	Server restarts continuously; Continuously unstable system

of this incident, and the other is that its anomaly pattern should satisfy the physical meaning and engineers' demand.

III. Approach

Motivated by the insights learned from the preliminary study, we propose an effective approach to identifying root-cause metrics named *PatternMatcher*. As presented in Fig. 3, *PatternMatcher* contains three components: coarse-grained anomaly detection, anomaly pattern classification, and root-cause metric ranking. Specifically, when an incident occurs in the online service system, *PatternMatcher* would be triggered, and related metrics to this incident would be collected based on system topology. First, *PatternMatcher* utilizes two-sample hypothesis test as the coarse-grained anomaly detection algorithm because of its high efficiency and accuracy. It can filter out the metrics performing normally during the occurrence of this incident so that the search space could be significantly reduced. Afterwards, we propose a novel anomaly pattern classification approach based on 1-D CNN [8], to further analyze the specific anomaly patterns of abnormal metrics. The goal is to filter out abnormal metrics whose anomaly patterns are not concerned by engineers, so as to improve the accuracy and provide more comprehensive information to engineers. Finally, we design a ranking strategy taking both

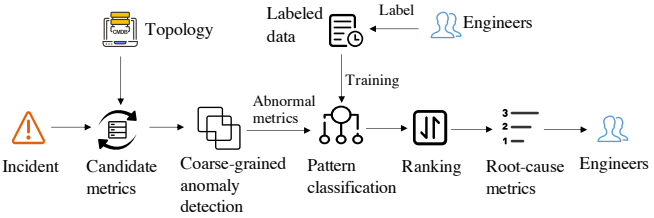


Fig. 3: Overview of *PatternMatcher*

the above two steps into consideration, so that engineers could check suspicious metrics according to the ranking list. In the following, we will present three main modules in detail.

A. Coarse-grained Anomaly Detection

As introduced in §II-C, one requirement of root-cause metrics is that they should behave abnormally when an incident occurs. In the literature, several approaches have been proposed to identifying root-cause metrics via anomaly detection. For example, FluxRank [4] adopts Kernel Density Estimation (KDE) to detect the change of metric distribution using the local data before the incident occurs. However, KDE suffers from several limitations in practice. First, the selection of kernel functions (e.g., RBF and Gaussian) and the bandwidth strongly affect the accuracy [11]. Besides, it is difficult for

KDE to accurately estimate the distribution given limited data points. Second, it is difficult to select an appropriate threshold to determine whether the metric behaves abnormally because the anomaly scores provided by KDE have no range limit.

To overcome the drawbacks of KDE, inspired by existing works [5], [6], we leverage two-sample hypothesis test as our coarse-grained anomaly detection algorithm, mainly due to its efficiency and effectiveness given numerous candidate metrics and short-term historical data. Besides, it is relatively simple to choose an appropriate threshold based on the output of p-value. In detail, given the occurrence time of an incident t and a metric m, we mainly focus on whether the metric is abnormal before time t. We compare the data between $m[t-l_1:t]$ (testing window S_A) and $m[t-l_1-l_2:t-l_1]$ (normal window S_N) using KS-test (Kolmogorov-Smirnov test [12]). Considering the fact that using too many data points for anomaly detection could result in system overhead in real deployment and few data points may undermine the accuracy, we set l_1 and l_2 to 10 and 30 data points based on the interactions with engineers. The hypothesis of the two-sample test can be formulated as follows:

$$\begin{cases} H_0: & S_A = S_N \\ H_1: & S_A \neq S_N \end{cases} \tag{1}$$

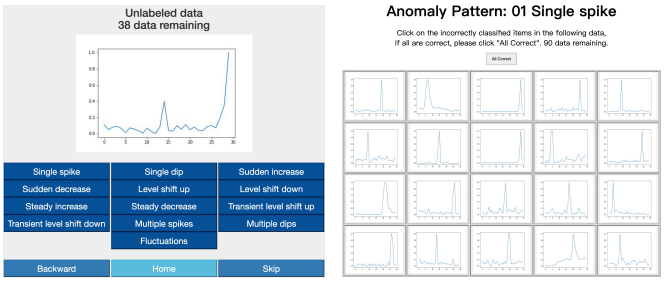
The two-sample test would output a p-value (P), indicating the confidence level. Given a threshold α (set to 0.05 in our experiments), if $P < \alpha$, we would reject H_0 and conclude S_A and S_N from different distributions. That is, this metric is abnormal and may be related to the incident. Otherwise, we will accept H_0, indicating the metric is normal and we can remove it from our candidate in the next steps.

B. Anomaly Pattern Classification

As presented in §II, anomaly patterns should be well considered to enhance accuracy and interpretability. Therefore, after identifying abnormal metrics, we need to further figure out the specific anomaly patterns of this metric, aiming to filter out the patterns that engineers do not care about. As shown in Fig. 2, we summarized 13 typical anomaly patterns via carefully analyzing large-scale real-world metrics. The goal of this step is to classify the anomalies into correct patterns.

1) Data Collection: The first step to solve the multi-class classification problem is to collect data with high-quality labels to train the model. As stated in §III-A, if the testing window S_A performs abnormally, we will focus on the anomaly pattern of $m[t-w:t]$ (S_P) and w is set to 30 data points. The length of the window is larger than S_A with length l_1 because we need to combine contextual information to identify anomaly patterns more precisely. Based on historical incidents and related metrics, we collected 1921 abnormal segments as our datasets. We split the training set (used to train the model), validation set (used to tune the hyperparameters), and testing set (used for evaluation) with the ratio of 6:2:2.

2) Labeling with Active Learning: Manually labeling thousands of segments is a labor-intensive and tedious task. To address the challenge, we draw support from active learning



(a) Single data annotation (b) Batch data confirmation

Fig. 4: Interface of labeling tool

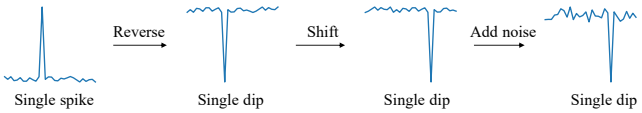


Fig. 5: Illustration of data augmentation

to reduce labeling efforts. Specifically, we first label 100 segments manually using our developed labeling tool (Fig. 4(a)) and then construct a simple model $\mathcal{M}$ for a rough pre-classification based on KNN [13] with DTW (Dynamic Time Warping) distance [14]. Next, we use model $\mathcal{M}$ to classify the remaining samples. If the results of top-k (k equals 3) are consistent, we tend to believe that the classification results are reliable. Otherwise, this segment is hard to be distinguished by the algorithm and should be labeled manually with the labeling tool. For the segments which have been well classified by $\mathcal{M}$, we develop a user interface (UI) for engineers to conveniently confirm the labels, as shown in Fig. 4(b). Finally, we need to label only less than 10% of the overall datasets (costing about 15 minutes), markedly reducing manual efforts.

3) Data Augmentation: Although we have obtained sufficient training data through the above step, the number of different patterns is imbalanced in our dataset. For example, the percentage of single spike is 11.35% (218 samples), while the percentage of steady increase is only 3.70% (71 samples), which could degrade the performance and robustness of the classifier. To address this problem, we leverage the technique of data augmentation [15] in *PatternMatcher*, which has been widely applied in the field of computer vision.

As shown in Fig. 5, we adopt three strategies to enrich our datasets. 1) *Reverse.* Two types of reverse can be applied in *PatternMatcher*, up to down (e.g., single spike to single dip) and left to right (e.g., level shift up to level shift down). 2) *Shift.* We can shift the segment by using a small step, which does not change its anomaly pattern. For example, a single spike pattern can be shifted to a new single spike where the positions of spikes are different. 3) *Adding Gaussian noises.* We randomly add Gaussian noises following the distribution of $\mathcal{N} \sim (0, 0.05)$ to the raw data. Through the above three methods, we obtain a high-quality training set, and then use it to train the classification model. The necessity of data augmentation will be demonstrated in §IV-E.

4) Classifier: Time series classification has been extensively investigated in the literature [16]. Existing works can

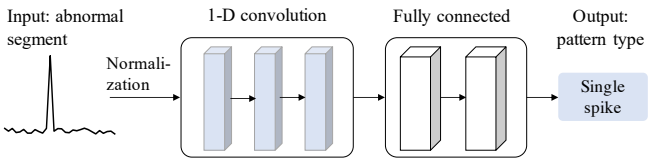


Fig. 6: Detailed network structure of our classifier

be divided into two categories: traditional machine learning based methods (e.g., KNN [17], Random Forest [18]) and deep learning based methods (e.g., CNN [19]). Based on our observation, feature-based traditional methods suffer from a significant challenge, i.e., requiring powerful features to train the classification model. The features of time series, however, are diverse (e.g., mean, standard deviation and trend), and it is a daunting task to extract and select useful features. KNN (distance-based method) is a lazy-learning method and suffers from extremely high time complexity. In comparison, deep learning based methods have the ability to learn features automatically. Specifically, we leverage the popular model 1-D CNN to extract features from time series and MLP (Multi-Layer Perceptron) as classifier [20]. The major strength of 1-D CNN is it performs well on time series data without any pre-determined transformation such as fast Fourier transform (FFT) and discrete wavelet transform (DWT), hand-crafted feature extraction and selection [21]. Besides, the architecture of 1-D CNN is compact compared with complex deep networks (e.g., VGGNet [22]), making it efficient under massive data. The detailed network structure of our classifier is displayed in Fig. 6. For the input of abnormal segment, we first adopt Min-max normalization to normalize its value to the range of [0,1], to eliminate the effect of amplitude and focus on the segment shape. The network is composed of three 1-D convolution layers (kernel size of all three layers is 5, output channel sizes are set to 64, 128, and 256) and two fully connected layers (output sizes are set to 64 and 13). The final output is computed by LogSoftmax. The loss function and optimizer are cross-entropy and Adam, respectively. The effectiveness of our classification model will be illustrated in §IV-E by compared with traditional approaches.

During incident diagnosis, if anomalies are detected in the testing window S_A ($m[t-l_1:t]$) of a metric, *PatternMatcher* would classify the segment S_P ($m[t-w:t]$) to identify its pattern. If engineers are not concerned about its pattern, this metric would be excluded. Otherwise, this metric would be included in the final ranking. Engineers' preferences for patterns can be configured in advance. In our experiments (§IV-B), we only configure some basic settings based on prior knowledge (e.g., filtering out all increase patterns of CPU usage and all decrease patterns of free memory space). In practical usage, engineers could add new settings based on their real demand to further improve the performance.

C. Root-cause Metric Ranking

Through the above two steps, we have filtered out the normal metrics and abnormal metrics with unconcerned anomaly patterns. Then, we provide a root-cause metric ranking list so that engineers could investigate suspicious components according to our recommended results.

As for the coarse-grained anomaly detection, it is intuitive that we should pay more attention to more abnormal metrics with larger anomaly scores (p-value). For the second step of pattern classification, as presented in §II-B3, different patterns exhibit different physical meanings with different severity levels. For example, the level shift up is much more severe than a single spike since the former is still in an abnormal state, and the latter has already recovered to normal. Therefore, the ranking strategy should take both the anomaly scores in the first step and severity levels of different patterns in the second step into consideration.

Specifically, we first acquire a basic anomaly score based on the p-value (P) for each metric, which is computed by $-\log\max(P, 0.0001)$. The goal of using max function is to constrain the scale of anomaly score. The threshold 0.0001 means we have 99.99% confidence being right. Then, we set weights for different anomaly patterns according to domain knowledge or engineers' demand. For example, we could assign Type-1 patterns in Table I with larger weights than those of Type-2 ones. In our experiements (§IV), we set the weights of Type-1 and Type-2 patterns to 0.8 and 0.2, respectively. We could obtain a promising result even though using such simple weights, and engineers can set weights more carefully based on real demand in practice. Finally, the root-cause score of each metric is computed as the following:

$$\text{rank\_score} = (-\log\max(P, 0.0001)) * pw \tag{2}$$

where pw is the weight of the corresponding anomaly pattern of this metric. Based on the output ranking list, the incident could be assigned to corresponding teams for further diagnosis.

IV. Evaluation

We conducted an extensive experimental study based on real-world data to demonstrate the effectiveness of *PatternMatcher*, aiming to answer the following research questions (RQs):

- RQ1: How does *PatternMatcher* perform in identifying root-cause metrics?
- RQ2: Do main components in *PatternMatcher* contribute to the overall performance?
- RQ3: How efficient is *PatternMatcher*?
- RQ4: How effective is our anomaly pattern classifier?

A. Dataset and Measurement

1) Dataset: To evaluate the performance of identifying root-cause metrics (RQ1, RQ2 and RQ3), we collected four datasets (113 incident cases in total) from different systems in a large commercial bank which supports hundreds of millions of users and contains hundreds of service systems. Table II presents the details of the datasets used in our experimental study. All metrics are sampled every minute. Dataset $\mathcal{A}$ and $\mathcal{B}$ are collected from two business systems, and the incidents are closely related to key business metrics (e.g., response time and success rate). Dataset $\mathcal{C}$ and $\mathcal{D}$ come from the database

TABLE II: Details of our experimental datasets

Dataset	Incident	Example candidate metrics	#Metrics	#Cases
$\mathcal{A}$	Application incident (low success rate, high response time, low transaction volume)	Database-related (oracle.session.active.total, oracle.lock.wait); middleware-related (weblogic.thread.pending, weblogic.webapp.sessions); network-related (system.net.bytes_sent, system.net.packets_in.count); server-related (system.cpu.pct_usage, system.io.await, system.mem.used); storage-related (san.read.time)	5213	16
$\mathcal{B}$	Application incident (high response time, high #user failures)	Network-related (system.net.bytes_sent, system.net.packets_in.count); resource-related (system.cpu.pct_usage, system.mem.used); middleware-related (weblogic.thread.pending, weblogic.webapp.sessions)	3229	19
$\mathcal{C}$	Database incident (high #active sessions, high CPU usage)	CPU-related (system.cpu.pct_usage, system.cpu.iowait); read/write-related (oracle.read.sequential, oracle.log.write.parallel); lock-related (oracle.lock.wait)	265	45
$\mathcal{D}$	Storage incident (high response time, disk failure)	I/O-related (san.write.pending, san.io.size; database-related (oracle.read.sequential, oracle.log.write.parallel); disk-related (san.disk.used, san.disk.free)	1060	33

and storage teams, which are responsible for all database and storage systems in the bank, respectively. The incidents of $\mathcal{C}$ and $\mathcal{D}$ may not directly destroy service quality, but engineers are very concerned about the anomalies of the key metrics (e.g., #active sessions in database and IOPS in storage). All of these incidents are collected from historical incident tickets and critical alerts. The root-cause metrics of each incident are obtained from historical troubleshooting records, which have been confirmed by experienced engineers.

2) Measurement: Following existing work [7], [23], we adopt $AC@k$ (top-k accuracy) and $Avg@k$ (top-k average accuracy) as measurements. $AC@k$ means the percentage of true root-cause metrics in the top-k result ($k = \{1, 3\}$). A higher value of $AC@k$, especially when the value of k is small, indicates that the approach could identify root-cause metrics more accurately. $Avg@k$ is the average performance of $AC@k$, computing by $\frac{1}{k}\sum_{i=1}^{k} AC@i$. As for the evaluation of anomaly pattern classification (RQ4), we adopt the widely-used classification metrics, i.e., precision, recall, and F1-score [24]. Considering multi-classification in our problem, we use the macro average to calculate these measurements [25].

B. RQ1: Overall Performance of PatternMatcher

To answer this RQ, we compare *PatternMatcher* with several existing approaches to demonstrate its effectiveness of root-cause metric identification.

- *FluxRank* [4] uses KDE to detect abnormal metrics and ranks root-cause metrics by their anomaly scores. The kernel function used in experiments is Gaussian.
- *ϵ-diagnosis* [5] adopts ϵ-statistical test based on energy distance to identify abnormal metrics. Due to the lack of ranking component in [5], we rank all abnormal metrics using p-value in our experiments.
- *Correlating Events with Time Series (CETS)* [6] utilizes nearest neighbor statistic to calculate the correlation between an incident and candidate metrics. The root-cause metrics are sorted based on the correlation scores.
- *MicroCause* [7] uses PC algorithm [7] to construct a causal graph on metrics, and conducts random walk to identify the root-cause metrics. It ranks root-cause metrics combining random walk visit times and anomaly scores.

Table III shows the results of *PatternMatcher* and compared approaches. From this table, we find *PatternMatcher* outperforms all four baselines in terms of all measurements. The $Avg@3$ of *PatternMatcher* achieves 0.96, 0.98, 0.88 and 0.80 on four datasets. The promising performance of *PatternMatcher* means that engineers could identify root-cause metrics more accurately and assist incident diagnosis rapidly under the help of *PatternMatcher*. In particular, an interesting observation is that *PatternMatcher* performs better on application incidents ($\mathcal{A}$ and $\mathcal{B}$). This is because candidate metrics of application incidents come from different infrastructure components and the root-cause metrics are relatively easy to identify. In contrast, candidate metrics in $\mathcal{C}$ and $\mathcal{D}$ are closely related to database and storage, and there are many abnormal metrics, incurring challenges for identifying root-cause metrics. Thus, the results demonstrate *PatternMatcher* could be well applied to application incidents triage.

We further analyzed the reasons why these baselines perform poorly. About FluxRank and ϵ-diagnosis, one common reason is the natural limitation of their anomaly detection algorithms. Specifically, the selection of kernel functions and bandwidths has a significant effect on the performance of KDE used in FluxRank [11], and it is time-consuming and challenging for engineers to choose optimal parameters manually. ϵ-statistics adopted in ϵ-diagnosis is designed specifically for long-tail latency problem, which is inappropriate in our scenario. Besides, ϵ-statistics relies on relative long-term data, which is impractical in the real world due to high I/O overhead under numerous. In terms of CETS, it relies on sufficient historical incident cases to analyze the correlation between an incident and all metrics. Besides, it is more accurate to identify root-cause metrics based on the current system behavior rather than static correlation score obtained from historical data. About MicroCause, it localizes root-cause metrics based on causality analysis with PC algorithm using long-term historical data. Besides, PC works as a black-box, and the output causal graph is hard to understand and evaluate [26]. More importantly, a major drawback in these baselines is that they ignore

TABLE III: Comparison between *PatternMatcher* and baseline approaches

Dataset	$\mathcal{A}$			$\mathcal{B}$			$\mathcal{C}$			$\mathcal{D}$			Average
Approach	$AC@1$	$AC@3$	$Avg@3$	$AC@1$	$AC@3$	$Avg@3$	$AC@1$	$AC@3$	$Avg@3$	$AC@1$	$AC@3$	$Avg@3$	$Avg@3$
PatternMatcher	**0.88**	**1.00**	**0.96**	**0.95**	**1.00**	**0.98**	**0.82**	**0.93**	**0.88**	**0.76**	**0.82**	**0.80**	**0.91**
FluxRank	0.69	0.81	0.75	0.63	0.74	0.68	0.64	0.76	0.69	0.64	0.73	0.69	0.70
ϵ-diagnosis	0.44	0.62	0.52	0.37	0.47	0.44	0.37	0.47	0.41	0.33	0.45	0.40	0.44
CETS	0.62	0.75	0.67	0.58	0.79	0.67	0.62	0.71	0.66	0.58	0.70	0.64	0.66
MicroCause	0.44	0.62	0.54	0.26	0.37	0.32	0.27	0.42	0.37	0.30	0.39	0.35	0.40
W/o APC	0.69	0.75	0.73	0.68	0.74	0.72	0.60	0.73	0.67	0.61	0.70	0.66	0.70
Raw ranking	0.81	0.94	0.88	0.90	0.95	0.93	0.76	0.91	0.84	0.70	0.79	0.75	0.85

the physical meanings of metrics and anomaly behaviors, leading to some false positives. As demonstrated in §II-B, despite the fact that some metrics perform abnormally before the incident occurs, the anomaly patterns are unimportant and could be neglected (for example, the decrease in CPU usage), and different anomaly patterns tend to show different explanatory power to root causes. Therefore, anomaly patterns should be incorporated in identifying root causes, which is the core reason why *PatternMatcher* outperforms baseline approaches.

Overall, *PatternMatcher* is able to identify root-cause metrics more accurately compared with four baseline approaches, providing clues for engineers in incident diagnosis.

C. RQ2: Contributions of Main Components

To investigate the contributions of two main components (i.e., anomaly pattern classification and root-cause metric ranking) to the overall performance of *PatternMatcher*, we construct two variants of *PatternMatcher* accordingly:

- W/o APC. We directly use two-sample test in §III-A to detect abnormal metrics and provide a ranking list based on p-value, without anomaly patterns classification (APC).
- Raw ranking. We replace our designed ranking strategy (§III-C) with raw ranking method using p-value, and other components remain unchanged, to illustrate the severity levels of different patterns.

The comparison results are presented in Table III. We find that the two variants of *PatternMatcher* indeed perform worse than *PatternMatcher* in terms of all the measurements in general. In particular, the average $Avg@3$ drops from 0.91 to 0.70 and 0.85 after removing anomaly pattern classification and adopting a raw ranking method without considering anomaly patterns. The results demonstrate the contributions of two main components to the overall performance. Specifically, incorporating anomaly pattern classification and considering physical meaning could filter out some abnormal but unconcerned patterns (e.g., the decrease of CPU usage) to enhance the accuracy and interpretability. Besides, different anomaly patterns may indicate different severity levels and show different explanatory power to root causes (§II-B), which should be paid extra attention in the ranking component.

D. RQ3: Time Efficiency

Intuitively, the task of identifying root-cause metrics requires high time efficiency to ensure short TTM (Time to Mitigate). If engineers need to wait for a long time to obtain the results, the service quality would be damaged. The time cost comparison on each dataset of *PatternMatcher* and baseline approaches are presented in Table IV. Clearly, given thousands of candidate metrics, *PatternMatcher* could identify root-cause metrics rapidly, less than 22 seconds on all datasets. Considering that *PatternMatcher* contains an anomaly pattern classifier, the overall running time is a little slower than FluxRank, ϵ-diagnosis and MicorCause. CETS suffers high time complexity since it relies on massive historical data and adopts time-intensive algorithms (KNN and DTW). Overall, *PatternMatcher* has satisfactory running time when dealing with thousands of metrics, indicating that *PatternMatcher* is indeed practical in the real world.

TABLE IV: Time cost (seconds) comparison between *PatternMatcher* and baseline approaches

Dataset	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$
#Candidate metrics	5213	3229	265	1060
PatternMatcher	21.58	13.12	1.45	4.18
FluxRank	15.63	9.31	1.21	2.95
ϵ-diagnosis	14.38	9.08	1.09	2.65
CETS	124.85	87.86	6.93	19.89
MicroCause	18.94	11.78	1.38	3.77

E. RQ4: Performance of Our Anomaly Pattern Classification

As demonstrated in §II-B and §IV-C, anomaly pattern classification is a crucial module and contributes much to *PatternMatcher*. To show the superiority of our algorithm based on 1-D CNN, we compare it with the following four baseline approaches, and the evaluation datasets have been introduced in §III-B1.

- KNN [13]. We apply both Euclidean distance (ED) and Dynamic Time Warping (DTW) [14] as distance measurement to the KNN model, and the value of k is set to 5. The final results are given by majority voting of top-k.
- Random Forest (RF) [18]. We design some hand-crafted features inspired by tsfresh [27], which is an open-source for time series feature extraction. Equipping with feature selection, we finally adopt 32 features in total. These features are fed into RF model to construct a classification model.
- Multi-Layer Perceptron (MLP) [20]. Similar to RF, we replace the classification model with MLP.

Table V shows the comparison results. Obviously, 1-D CNN model outperforms the four compared models in terms of all

TABLE V: Precision (P), recall (R) and F1-score (F1) comparison of anomaly pattern classification between 1-D CNN and baseline approaches

Approach	With DA			Without DA		
	P	R	F1	P	R	F1
1-D CNN	**0.98**	**0.98**	**0.98**	**0.94**	**0.93**	**0.93**
KNN (ED)	0.83	0.84	0.83	0.79	0.81	0.78
KNN (DTW)	0.94	0.90	0.91	0.91	0.88	0.89
RF	0.94	0.92	0.93	0.92	0.91	0.91
MLP	0.92	0.91	0.92	0.90	0.89	0.89

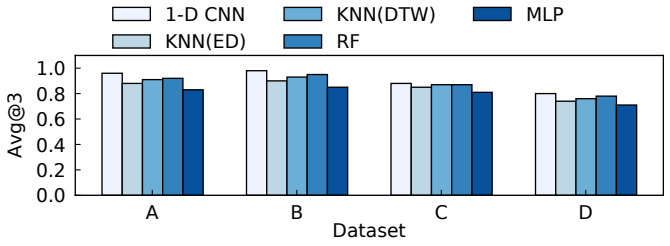


Fig. 7: Performance of *PatternMatcher* with different anomaly pattern classifers

the measurements, achieving the F1-score of 0.98. The F1-score of KNN with DTW distance without manual feature extraction is acceptable. However, KNN as a lazy learning method, suffers from high time cost for online deployment. About feature-based approaches (RF and MLP), it is labor-intensive to extract and select powerful features to characterize anomaly patterns manually. In contrast, 1-D CNN has the ability to automatically learn useful features without manual efforts. In summary, taking both classification performance and practical usage into consideration, our approach based on 1-D CNN is indeed effective. As stated in §III-B3, we leverage data augmentation (DA) to overcome the data imbalance problem and enhance model robustness. To prove its contribution, we compare the performance of classification models with DA and without DA. As shown in Table V, data augmentation indeed brings an improvement on performance. The F1-score of 1-D CNN would drop from 0.98 to 0.93 without DA, and so are other classification methods.

Besides, we further analyze the impact of different classification methods on the overall performance of *PatternMatcher*. Fig. 7 presents the $Avg@3$ on four datasets under different classifiers. Obviously, *PatternMatcher* using 1-D CNN method could identify root-cause metrics more accurately than compared approaches. Thus, the superiority of the classification model could directly affect the performance of identifying root-cause metrics.

V. Discussion

A. Deployment

We have successfully deployed *PatternMatcher* in a top commercial bank in the country. Fig. 8 shows the deployment architecture of *PatternMatcher*, including data collector, computation and front-end web service. Data collector aims to collect and store all metrics in the online service system. The monitors produce data to Kafka (an open-source

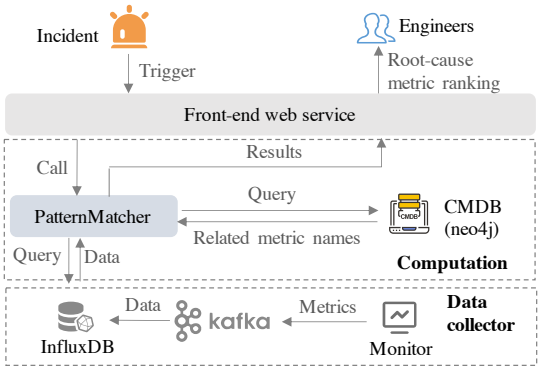


Fig. 8: Deployment architecture of *PatternMatcher*

distributed event streaming platform) [28], and a script consumes data from Kafka and writes to InfluxDB (a time-series database) [29] periodically. The computation component is supported by Apache Flink [30], which can process data with high performance and low latency in a distributed way. In practical usage, when an incident occurs, engineers would start to run *PatternMatcher* from the interface. *PatternMatcher* queries related metric names from CMDB (Configuration Management Database), which stores complex system dependencies using Neo4j [31]. Then, *PatternMatcher* reads corresponding metric data from InfluxDB and conducts the computation task. Finally, engineers could obtain a report generated by *PatternMatcher* from the front-end interface.

B. Success Story

Based on the real feedback, *PatternMatcher* has successfully assisted engineers in incident diagnosis. Engineers could investigate suspicious components based on the results given by *PatternMatcher*, so that the incident could be assigned to responsible teams correctly for further troubleshooting. In the following, we present two successful cases in practice.

1) Case I (Diagnose Database Incident): The database incident was identified by the increase in #active sessions, a key metric of Oracle database. The root cause is a burst of write operations. With traditional manual troubleshooting, engineers need to manually inspect 265 metrics related to database, which is extremely time-consuming. However, *PatternMatcher* could automatically check these metrics and three write-related metrics (log file parallel write, log file sync, db file parallel write) are ranked in the top-5 results accurately. We also analyzed the results of FluxRank, which incorrectly ranks CPU usage and read-related metrics (e.g., db file sequential read) in the top. In comparison, *PatternMatcher* could filter out such false positives via a more effective coarse-grained anomaly detection and anomaly pattern classification, because the decrease in CPU usage and read-related metrics are positive phenomenons and cannot be the root causes. As a result, *PatternMatcher* could provide accurate clues for further diagnosis and avoid wrong mitigation actions.

2) Case II (Diagnose Application Incident): Engineers detected the incident via the sharp increase in the number of failed transactions of a service. *PatternMatcher* rapidly checked thousands of infrastructure metrics supporting the

service and generated a report within 1 minute. CPU usage, memory usage, and heap space are top-3 root-cause metrics. CPU usage shows the pattern of multiple spikes, and memory usage and heap space are steadily increasing. Through this report, engineers infer that this incident may be induced by frequent fullGC operations, which is a CPU-intensive task and occupies too much heap space. Inspired by the clue, engineers further investigated the dump files and related logs, and finally confirmed this incident was caused by a piece of buggy code.

C. Lessons Learned

Root-cause metric is not equal to the final root cause. The goal of identifying root-cause metrics is to provide some clues and directions for further diagnosis. However, localizing the final root causes (e.g., buggy code, error configurations, and hardware fault) needs to integrate multiple data sources (e.g., logs and traces), which can be our future work.

PatternMatcher could also be applied in other tasks. In addition to identifying root-cause metrics, *PatternMatcher* could be triggered at any time based on engineers' demand to inspect the behaviors of numerous metrics rapidly and accurately. For example, engineers could leverage *PatternMatcher* for regular system inspection to identify some risky anomaly patterns (e.g., steady increase in memory usage), so that engineers could take some proactive actions to prevent service outages.

D. Limitation

One limitation of our work is the lack of detailed analysis of parameters (l_1, l_2 and w in §III) and anomaly pattern configurations (pw in §III-C) due to the space limit. In our experiments, we select the parameters based on the domain experience from engineers considering both accuracy and efficiency, and config pattern weights are based on some prior knowledge. In practical usage, engineers could tune these parameters and config weights according to their preferences. In the future, we will further analyze the impact of different parameters and configurations on the performance.

VI. Related Work

A. Root Cause Analysis

In the literature, some approaches have been proposed for identifying root-cause metrics. FluxRank [4] utilizes KDE to detect abnormal metrics and ranks root-cause metrics based on anomaly scores. ϵ-diagnosis [5] adopts ϵ-statistical test to identify suspicious metrics to reduce search space. Luo et al. [6] proposed to analyze the correlation between incidents and metrics using KNN-based statistical test. However, these works perform poorly in the real world due to the natural limitation of their algorithms and ignoring the physical meanings of metrics and corresponding anomaly patterns. The effectiveness of *PatternMatcher* compared with existing works has been demonstrated in §IV-B.

Besides, tremendous efforts have been devoted to root cause analysis using other technical routes, including graph-based approaches, log-based approaches and so on. Graph-based approaches [7], [32], [33], [34], [35] mainly aim to localize the root cause based on the graph and random walk. The graph characterizing the causality and call relationship between different data sources or components could be constructed by automated algorithms (e.g., PC [36]) or manual configuration (e.g., G-RCA [37]). Log-based localization aims to draw key clues from a large number of logs for incident diagnosis [38], [39], [40], [41], [42], but these approaches are not suitable for our scenarios and numerous logs incur high time complexity. The goal of our work is to utilize metric data to identify clues for incident diagnosis. Incorporating various data sources (e.g., topology and logs) to localize root causes more accurately can be our future work.

B. Time Series Anomaly Detection

Time series anomaly detection has been extensively studied in the literature [43], [44], [45], [46], [47]. For example, Donut [48] applies Variational Auto-Encoder (VAE) to detect anomalies on seasonal metrics. Ren et al. [49] proposed an anomaly detection approach based on Spectral Residual and CNN. However, the existing streaming algorithms require long-term training data and high training cost, which cannot be applied to our scenario under numerous candidate metrics. Besides, all existing works aiming to detect statistical anomalies fail to consider the anomaly patterns and hidden physical meanings, leading to poor interpretability and some false positives. To our best knowledge, we are the first to focus on anomaly patterns in the problem of root-cause metric identification and propose an effective anomaly pattern classification approach, to enhance the performance and interpretability.

VII. Conclusion

Identifying root-cause metrics accurately could provide clues for engineers to diagnose incidents. To better understand the problem, we conduct the first preliminary study to investigate the performance of current practice and characteristics of root-cause metrics. Inspired by the findings obtained from the study, we propose an effective approach named *PatternMatcher*, which contains three components, coarse-grained anomaly detection, anomaly pattern classification, and root-cause metric ranking. An extensive experimental study based on real-world data containing diverse incidents demonstrates the effectiveness of *PatternMatcher* and the $Avg@3$ accuracy on four datasets could achieve 0.91 on average, outperforming all baseline approaches. Furthermore, *PatternMatcher* has been deployed in a large commercial bank and we share two successful cases to show the practical usage.

ACKNOWLEDGMENT

We thank Xuanrun Wang and Bo Cheng for developing the labeling tool. This work is supported by the National Key Research and Development Program of China (Grant No.2019YFE0105500), the State Key Program of National Natural Science of China under Grant 62072264, and the Beijing National Research Center for Information Science and Technology (BNRist) key projects.

REFERENCES

[1] J. Jiang, W. Lu, J. Chen, Q. Lin, P. Zhao, Y. Kang, H. Zhang, Y. Xiong, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Virtual Event USA: ACM, Nov. 2020, pp. 1410–1420.

[2] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu *et al.*, "Towards intelligent incident management: why we need it and how we make it," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1487–1497.

[3] https://www.statista.com/statistics/753938/worldwide-enterprise-server-hourly-downtime-cost/, Average cost per hour of enterprise server downtime worldwide in 2019, [Online; accessed 04-May-2021].

[4] P. Liu, Y. Chen, X. Nie, J. Zhu, S. Zhang, K. Sui, M. Zhang, and D. Pei, "Fluxrank: A widely-deployable framework to automatically localizing root cause machines for software service failure mitigation," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2019, pp. 35–46.

[5] H. Shan, Y. Chen, H. Liu, Y. Zhang, X. Xiao, X. He, M. Li, and W. Ding, "?-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms," in *The World Wide Web Conference*, 2019, pp. 3215–3222.

[6] C. Luo, J.-G. Lou, Q. Lin, Q. Fu, R. Ding, D. Zhang, and Z. Wang, "Correlating events with time series for incident diagnosis," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1583–1592.

[7] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia, Z. Wang, and D. Pei, "Localizing failure root causes in a microservice through causality inference," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 2020, pp. 1–10.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[9] https://grafana.com/, Grafana, [Online; accessed 04-May-2021].

[10] https://www.elastic.co/kibana, Kibana, [Online; accessed 04-May-2021].

[11] A. Qahtan, S. Wang, and X. Zhang, "Kde-track: An efficient dynamic density estimator for data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 3, pp. 642–655, 2016.

[12] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

[13] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 4, pp. 325–327, 1976.

[14] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in *KDD workshop*, vol. 10, no. 16. Seattle, WA, USA:, 1994, pp. 359–370.

[15] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[16] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[17] Y.-H. Lee, C.-P. Wei, T.-H. Cheng, and C.-T. Yang, "Nearest-neighbor-based approach to time-series classification," *Decision Support Systems*, vol. 53, no. 1, pp. 207–217, 2012.

[18] M. Belgiu and L. Drăguţ, "Random forest in remote sensing: A review of applications and future directions," *ISPRS journal of photogrammetry and remote sensing*, vol. 114, pp. 24–31, 2016.

[19] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *Journal of Systems Engineering and Electronics*, vol. 28, no. 1, pp. 162–169, 2017.

[20] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[21] L. Eren, T. Ince, and S. Kiranyaz, "A generic intelligent bearing fault diagnosis system using compact adaptive 1d cnn classifier," *Journal of Signal Processing Systems*, vol. 91, no. 2, pp. 179–189, 2019.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

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

[24] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[25] M. Cissé, M. Al-Shedivat, and S. Bengio, "Adios: Architectures deep in output space," in *International Conference on Machine Learning*. PMLR, 2016, pp. 2770–2779.

[26] P. Spirtes, C. N. Glymour, R. Scheines, and D. Heckerman, *Causation, prediction, and search*. MIT press, 2000.

[27] https://tsfresh.readthedocs.io/en/latest/#, tsfresh, [Online; accessed 21-Apr-2021].

[28] https://kafka.apache.org/, Kafka, [Online; accessed 21-Apr-2021].

[29] https://www.influxdata.com/, InfluxDB, [Online; accessed 21-Apr-2021].

[30] https://flink.apache.org/, Flink, Scalable Stream and Batch Data Processing, [Online; accessed 04-May-2021].

[31] https://neo4j.com/, neo4j, [Online; accessed 21-Apr-2021].

[32] 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.

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

[34] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, and et al., "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *ESEC/FSE*. ACM, 2019, pp. 683–694.

[35] D. Liu, C. He, X. Peng, F. Lin, C. Zhang, S. Gong, Z. Li, J. Ou, and Z. Wu, "Microhecl: High-efficient root cause localization in large-scale microservice systems," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2021, pp. 338–347.

[36] J. Runge, P. Nowack, M. Kretschmer, S. Flaxman, and D. Sejdinovic, "Detecting and quantifying causal associations in large nonlinear time series datasets," *Science Advances*, vol. 5, no. 11, p. eaau4996, 2019.

[37] H. Yan, L. Breslau, Z. Ge, D. Massey, D. Pei, and J. Yates, "G-rca: a generic root cause analysis platform for service quality management in large ip networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1734–1747, 2012.

[38] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 60–70.

[39] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 102–111.

[40] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298.

[41] B. C. Tak, S. Tao, L. Yang, C. Zhu, and Y. Ruan, "Logan: Problem diagnosis in the cloud using log-based reference models," in *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2016, pp. 62–67.

[42] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *arXiv preprint arXiv:2009.07237*, 2020.

[43] 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*, 2015, pp. 1939–1947.

[44] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*, vol. 89. Presses universitaires de Louvain, 2015, pp. 89–94.

[45] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed, "Deepant: A deep learning approach for unsupervised anomaly detection in time series," *IEEE Access*, vol. 7, pp. 1991–2005, 2018.

[46] X. Zhang, J. Kim, Q. Lin, K. Lim, S. O. Kanaujia, Y. Xu, K. Jamieson, A. Albarghouthi, S. Qin, M. J. Freedman *et al.*, "Cross-dataset time series anomaly detection for cloud systems," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019, pp. 1063–1076.

[47] M. Braei and S. Wagner, "Anomaly detection in univariate time-series: A survey on the state-of-the-art," *arXiv preprint arXiv:2004.00433*, 2020.

[48] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 187–196.

[49] 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 & Data Mining*, 2019, pp. 3009–3017.