Understanding and Handling Alert Storm for Online Service Systems

Nengwen Zhao*
Tsinghua University; BNRist

Junjie Chen†
Tianjin University, College of Intelligence and Computing

Xiao Peng
China EverBright Bank

Honglin Wang, Xinya Wu
Yuanzong Zhang
BizSeer

Zikai Chen
Tsinghua University; BNRist

Xiangzhong Zheng
BizSeer

Xiaohui Nie
Tsinghua University; BNRist

Gang Wang, Yong Wu
Fang Zhou
China EverBright Bank

Wenchi Zhang, Kaixin Sui
BizSeer

Dan Pei
Tsinghua University; BNRist

ABSTRACT

Alert is a kind of key data source in monitoring system for online service systems, which is used to record the anomalies in service components and report to engineers. In general, the occurrence of a service failure tends to be along with a large number of alerts, which is called **alert storm**. However, alert storm brings great challenges to diagnose the failure, because it is time-consuming and tedious for engineers to investigate such an overwhelming number of alerts manually. To help understand alert storm in practice, we conduct the first empirical study of alert storm based on large-scale real-world alert data and gain some valuable insights. Based on the findings obtained from the study, we propose a novel approach to handling alert storm. Specifically, this approach includes alert storm detection which aims to identify alert storm accurately, and alert storm summary which aims to recommend a small set of representative alerts to engineers for failure diagnosis. Our experimental study on real-world dataset demonstrates that our alert storm detection can achieve high F1-score (larger than 0.9). Besides, our alert storm summary can reduce the number of alerts that need to be examined by more than 98% and discover representative alerts accurately. We have successfully applied our approach to the service maintenance of a large commercial bank (China EverBright Bank), and we also share our success stories and lessons learned in industry.

*BNRist: Beijing National Research Center for Information Science and Technology
†Junjie Chen is the corresponding author.

ICSE-SEIP '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7123-0/20/05.
https://doi.org/10.1145/3377813.3381363

CCS CONCEPTS

• **Software and its engineering** → *Maintaining software.*

KEYWORDS

Alert Storm; Alert Summary; Problem Identification; Failure Diagnoisis

ACM Reference Format:
Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, Yong Wu, Fang Zhou, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Understanding and Handling Alert Storm for Online Service Systems. In *Software Engineering in Practice (ICSE-SEIP '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3377813.3381363

1 INTRODUCTION

Online service systems, such as online banking and search engine, have become indispensable parts in our daily life. However, due to the large scale and complexity of these systems, service failures are inevitable in practice [12]. More specifically, service failures can be caused by many factors, such as hardware outages, software bugs and unexpected user load changes [25, 26]. The service failures could cause slow response, unavailability, violation of service level agreements (SLA), ultimately customer dissatisfaction, and huge economic loss. For example, according to a study conducted on 63 data center organizations in the U.S, the average cost of downtime has steadily increased from $505,502 in 2010 to $740,357 in 2016 [1] [4]. Therefore, detecting failures accurately and diagnosing failures quickly are quite vital for online service systems.

Currently, to ensure the service quality, engineers collect various types of monitoring data, including Key Performance Indicators (KPIs) [21, 23, 24], logs [9, 12], traces [26] and incidents [4, 5], and then check these data by manually defining many rules. Once these data violate the pre-defined rules (e.g., the latency of a request exceeds a pre-defined threshold), alerts would be produced to notify

[1] https://www.ponemon.org/blog/2016-cost-of-data-center-outages.

On-Call Engineers (OCEs) and then engineers would start diagnosis for the system according to the alerts. In this way, a potential service failure could be noticed and mitigated earlier.

However, in practice, the occurrence of a service failure tends to be along with a large number of alerts (e.g., thousands of alerts produced every minute) rather than one or two alerts. This is because an online service system consists of a large number of components, each of which has a huge amount of monitoring data, and different components tend to be affected by each other. We call this phenomenon **alert storm**. Faced with such an overwhelming number of alerts, it is very time-consuming and tedious for engineers to manually examine each alert to identify and diagnose the real failure [10]. That is, the ideal goal of using alerts to help ensure the service quality is good, but it suffers from the great challenge in reality due to the existence of alert storm. Therefore, to facilitate the assurance of the service quality, understanding and handling alert storm are very essential.

To date, there is no work investigating alert storm. To help understand alert storm, we conducted the first empirical study to investigate it. More specifically, we studied alert storm based on a large amount of real-world alert data from a large commercial bank (China EverBright Bank). Through this study, we obtained three key findings: (1) Alert storm occurs frequently (about once a week) and brings great trouble to engineers in practice, i.e., taking several engineers about an hour to deal with alert storm on average. (2) The current practice of identifying alert storm is just to set a fixed threshold manually, which cannot fit the dynamic online service environment and thus could lead to poor performance. (3) Some alerts in alert storm are regular, which are irrelevant to the service failure, and also many alerts relevant to the failure have certain correlation, i.e., textual and topological correlation. To sum up, on the one hand, this study further motivates the necessity of handling alert storm; on the other hand, it provides some guidelines to help us handle alert storm.

Inspired by the findings from our empirical study, in this paper we novelly propose an approach to handling alert storm, which first accurately identifies alert storm (**alert storm detection**) and then effectively selects a few representative alerts from alert storm recommending to engineers instead of analyzing all the alerts (**alert storm summary**). More specifically, in the stage of alert storm detection, instead of manually setting a fixed threshold, we formulate the problem of alert storm detection as online change point detection and then adopt Extreme Value Theory (EVT) [6, 20] to detect alert storm adaptively and accurately. In the stage of alert storm summary, the goal is to select a set of alerts that are relevant to the service failure and are able to reflect the failure from diverse aspects. With this intuition, we first design an alert denoising method to filter irrelevant alerts by learning alert patterns in normal states of the system. Then, we discriminate alerts reflecting the service failure from different aspects into different groups by clustering based on their textual and topological correlation. Finally, we select the most representative alert from each cluster to form a small set of alerts for investigation instead of examining all the alerts in alert storm, which can largely save engineers' effort.

To evaluate the effectiveness of our proposed approach, we conducted an experimental study based on large-scale real-world alert data from a large commercial bank and collected 166 alert storm cases based on historical failure tickets. The experimental results demonstrated that our approach can detect alert storm more accurately compared with the traditional threshold-based method, achieving the F1-score of larger than 0.9. Also, our approach can reduce the number of alerts that engineers need to examine by larger than 98% and accurately recommend representative alerts, which indeed facilitates engineers to diagnose the service failures. In addition, we also evaluated the two main steps (i.e., alert denoising and alert discrimination) in alert storm summary, demonstrating that our methods perform better than baseline methods. In particular, we have successfully deployed our approach in the large commercial bank and shared some experience from practice.

To sum up, this work has the following main contributions:

- We conducted the first empirical study to understand alert storm based on real-world alert data from a large commercial bank, delivering a series of valuable findings.
- We proposed the first approach to handling alert storm, consisting of alert storm detection and alert storm summary. The former aims to adaptively and accurately detect alert storm based on Extreme Value Theory [6, 20], while the latter aims to select a small set of representative alerts for investigation by proposing an alert denoising method and an alert discrimination method.
- We conducted an experimental study to evaluate the effectiveness of our approach based on real-world alert data, demonstrating the great effectiveness of our approach for both alert storm detection and alert storm summary.

2 AN EMPIRICAL STUDY ON ALERT STORM

To help understand alert storm, in this section we present the first large-scale empirical study on alert storm. In this study, we address the following research questions:

- **RQ1**: What is the distribution of the number of alerts?
- **RQ2**: What is the cost of handling alert storm?
- **RQ3**: How do engineers think about alert storm in industry?
- **RQ4**: What are the characteristics of alert storm by case analysis?

Our study is performed based on the large-scale real-world alert data provided by a large commercial bank (China EverBright Bank), and all the alerts are generated from 2016/07/01 to 2019/06/31 (three years in total) through the unified alert management platform. In total, the number of alerts is up to 3 millions. Also, this bank has hundreds of services and thousands of servers, which support more than one hundred million users.

2.1 Alert Distribution

To understand the alert distribution, we used the 3-year-long alert data and counted the number of alerts per minute. Figure 1 presents the number of occurrences under different scale of (#Alerts per minute). From this figure, we observed in about 96.87% cases the number of alerts per minute is no more than 10, and in about 99.83% cases the number of alerts per minute is no more than 100, which are normal states of the online service system. However, in about 0.17% cases, hundreds of to thousands of alerts occur in one minute. In these cases, the number of alerts is so overwhelming that engineers cannot properly handle them, which are so-called alert storm.

In practice, engineers adopt the fixed threshold strategy to identify alert storm, where the threshold is defined according to their

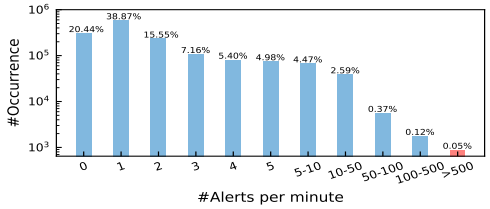
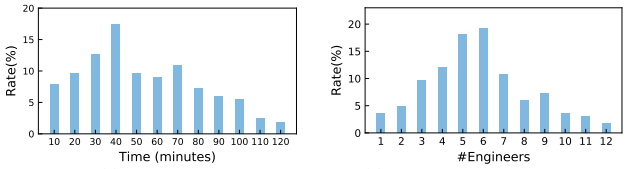


Figure 1: The number of occurrences under different scale of (#Alerts per minute)

experience. For example, in our collaborated bank, they define the cases that #(alerts per minute) is larger than 500 as alert storm. However, the pre-defined threshold may be out of operation in practice, since online service systems tend to be changed frequently, for example, the number of alerts could be increased due to the development of new services. At that time, the threshold should be also adjusted by considering the specific changes. Besides, the setting of the threshold varies among different engineers due to their different experience and also it is hard to determine whether the setting is reasonable enough in practice [10]. Therefore, that indicates the necessity of designing a method to adaptively and accurately identify alert storm.

2.2 Cost of Handling Alert Storm

We investigated the cost of handling alert storm in terms of the following two measurements: 1) the time cost, and 2) the number of engineers involved. More specifically, we collected 166 alert storm cases from the 3-year-long alert data according to historical failure tickets. From the detailed records in tickets, we get the time cost and the number of involved engineers of each case, whose results are shown in Figure 2. From this figure, we can observe it takes about 55 minutes and involves 6 engineers to handle an alert storm case on average. In particular, in some extreme cases, dealing with an alert storm case costs about two hours and involves more than 10 engineers, which occupies much effort and many resources, leading to negative influence on system development and maintenance. Therefore, an effective approach to assisting engineers to handle alert storm is in an urgent demand.



(a) Time cost (b) The number of engineers

Figure 2: The time spent on handling alert storm and the number of involved engineers

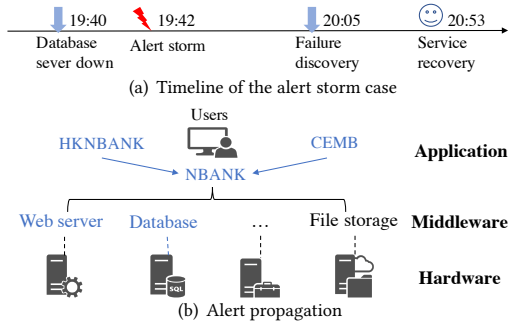
2.3 Survey on Alert Storm

To investigate the comments of engineers on alert storm in industry, we designed a questionnaire including a few questions, and invited some engineers from different service teams in the collaborated bank to take part in the survey. More specifically, we sent out 50 questionnaires in total and received 44 completed questionnaires.

By analyzing our collected questionnaires, we found that alert storm occurs frequently in industry, and the majority of engineers (86.3%) thought the frequency of alert storm is about once a week. Besides, nearly 88.6% of the 44 respondents argued the fixed threshold for alert storm detection is not very accurate and adaptive with some false positives. Therefore, an effective alert storm detection method is indeed desired. All the engineers involved in this survey agreed that alert storm is a troubling problem. The disturbing reasons included the number of alerts is too large to identify the problem manually (77.3%), messages and emails explosion (68.2%), and bad impact on normal work (72.7%). Almost everyone (97.7%) agreed it is very meaningful to reduce the number of alerts in alert storm, and about 70.5% of the engineers thought the maximum acceptable number of alerts per minute is 30. Therefore, the problem of alert storm is indeed a headache for engineers, and extracting a small set of representative alerts as the alert-storm summary seems to be a promising direction to relieve this problem.

2.4 Case Analysis

To understand the characteristics of alert storm, we manually analyzed some cases in depth and gained a series of insights. Here, we chose one typical case as the representative to present the characteristics of alert storm, which are also hold in the other cases. Figure 3(a) shows the timeline of this case. It was caused by a database server down at 19:40 and the alert storm occurred at 19:42 (the number of alerts at this minute is 459). However, this alert storm was not detected by the fixed threshold method (lower than the threshold 500). Finally, the failure was discovered by user complaint at 20:05 and engineers took 48 minutes to mitigate the failure. As shown in Figure 3(b), due to database disconnection, the service (NBANK) cannot receive data, which leads to a series of alerts generated by other components in NBANK. Subsequently, other services calling it also generated alerts. All the components generating alerts due to this failure labeled in blue color and the alert propagation relationships are labeled in blue line in Figure 3(b).



(a) Timeline of the alert storm case

(b) Alert propagation

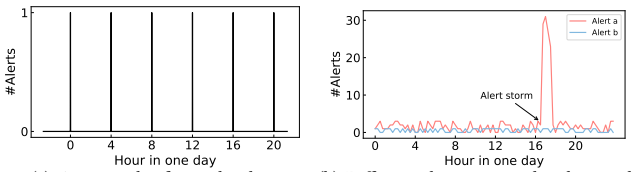
Figure 3: A typical alert storm case

We sampled some typical alerts from this case shown in Table 1. Based on our analyzed cases, we found that there are many redundant alerts in alert storm. In general, redundant alerts can be divided into the following two categories:

Regular alerts. Some alerts frequently occur no matter whether there is a failure or not. For example, as shown in Figure 4(a), periodic alerts regularly occur with a fixed time interval. Figure 4(b) further uses an intuitive example to explain the difference between the alerts relevant to the failure (Alert a) and regular alerts (Alert b). That is, the occurrence of regular alerts is not affected by alert storm. In Table 1, the regular alerts are labeled in gray color. Therefore, regular alerts are not helpful to failure diagnosis actually and sometimes even mislead engineers, so we should filter them out.

Table 1: An example to present some typical alerts in the alert storm case in Figure 3

No	Time	Content	Service	Server	Type	Severity
1	2018/4/7 19:40	Node Ping checks and alarms. **Packet loss 100%**.	NBANK	P12	Network	3
2	2018/4/7 19:42	The number of Oracle sessions is high.	NBANK	P12	Database	3
3	2018/4/7 19:42	CPU usage: Can't get necessary data.	NBANK	P12	OS	1
4	2018/4/7 19:42	Can't get Weblogic queue (NBANKAPP). **Timeout**.	NBANK	P4	Middleware	1
5	2018/4/7 19:42	CPU usage is 72%. (Threshold is 70%).	EPAY	P1	OS	3
6	2018/4/7 19:42	Batch automation agent port. **Connection refused**.	CEMB	P3	Application	1
7	2018/4/7 19:42	Weblogic port alarms. **Connection refused**.	HKNABNK	P6	Middleware	3
8	2018/4/7 19:42	Weblogic JDBC pool is not running. CEMB: MbankMgmt.	CEMB	P3	Middleware	2
9	2018/4/7 19:42	Weblogic JDBC pool is not running. CEMB: MClientServer.	CEMB	P7	Middleware	2
10	2018/4/7 19:42	Standby database recover-mode is MANAGED.	SCFP	P13	Database	3



(a) An example of periodic alert (b) Difference between regular alerts and anomalous alerts

Figure 4: Illustration of the regular alerts

Correlated alerts. As shown in Figure 3(b), the database server down can lead to many related alerts, and thus there are many correlated alerts in alert storm. In the study, we investigated the correlation between alerts from two aspects, i.e., text and topology. Textual correlation means that the textual contents of two alerts are similar, e.g., alerts 8 and 9 in Table 1. In terms of topological correlation, Figure 3(b) clearly presents the propagation of alerts in topology. More specifically, topological correlation includes two aspects: software (e.g., NBANK, CEMB and HKNBANK) and hardware (e.g., a switch failure may cause the related servers to generate alerts). These correlated alerts tend to reflect the failure from similar aspects, and thus grouping them can facilitate engineers' investigation to some degree.

In summary, through the empirical study including the analyzed typical alert storm cases, we obtain the following three key findings:

- Alert storm frequently occurs in development and maintenance of online service systems, and it takes several engineers about one hour on average to handle alert storm.
- The fixed threshold method used for alert storm detection is far from satisfying in practice.
- Some alerts in alert storm are irrelevant to the failure, and also many alerts relevant to the failure have certain correlation, i.e., textual and topological correlation.

Therefore, it is necessary to explore an effective approach to handling alert storm, including detecting alert storm accurately and adaptively, filtering out regular alerts, and grouping correlated alerts, in order to facilitate failure diagnosis from alert storm.

3 APPROACH

3.1 Overview

Inspired by the findings from our empirical study, we propose the first approach to handling alert storm and the overview of our approach is presented in Figure 5. Our approach contains two components, alert storm detection and alert storm summary. Specifically, for online alert stream, we leverages Extremely Value Theory (EVT) [20] to detect alert storm adaptively and accurately. Then, if an alert storm case is discovered, the second component will be triggered to assist engineers to identify the problem. Alert storm summary includes three steps: learning-based alert denoising, clustering-based alert discrimination, and representative alert selection. The overview of alert storm summary is depicted in Figure 6. In detail, alert denoising aims to filter out irrelevant alerts to the failure. Alert discrimination aims to divide the alerts into different groups via clustering based on their textual and topological correlation so as to reflect the failure from different aspects. Representative alert selection aims to extract the most representative alert from each cluster to recommend to engineers [12]. Through the above three steps, we can acquire a small set of representative alerts that are relevant to the failure and reflect the failure from diverse aspects, so that engineers can spend less time handling the alert storm instead of analyzing all the alerts.

3.2 Alert Storm Detection

In real world, the fixed threshold method (e.g., #alerts/minute > 500) is commonly used to identify alert storm. However, the designed threshold rules cannot accommodate to dynamic service systems. For example, new service deployment will lead to the increase of the number of alerts. Besides, different engineers have their own preference when setting threshold [10]. Therefore, to deal with alert storm better, we first need an adaptive and accurate alert storm detection method.

Technically, alert storm detection can be formulated as the online change point (spike) detection problem. Extreme Value Theory (EVT) is a popular statistical method dealing with the extreme deviations from the median of probability distributions, which has been applied in predicting the probability distribution of extreme floods, tornado outbreaks, and many other unusual events [6]. Motivated by EVT used for stream anomaly detection [20], we propose to apply EVT to identify alert storm. The goal of EVT is to find the law of extreme values, which are usually placed at the tails of a probability distribution. EVT does not require to hand-set thresholds and it makes no assumption on data distribution. Following [20], we adopt Peaks-Over-Threshold (POT) based on generalized Pareto distribution (GPD) to fit the tail of the distribution, and the parameters of GPD can be estimated by Maximum Likelihood Estimation (MLE). Due to the limit of space, we omit more details about EVT, which can be found in [6, 20].

In our approach, we use the normal number of alerts per minute in history to fit the POT model and apply the model to detect extreme value (alert storm) online. Most importantly, EVT has the

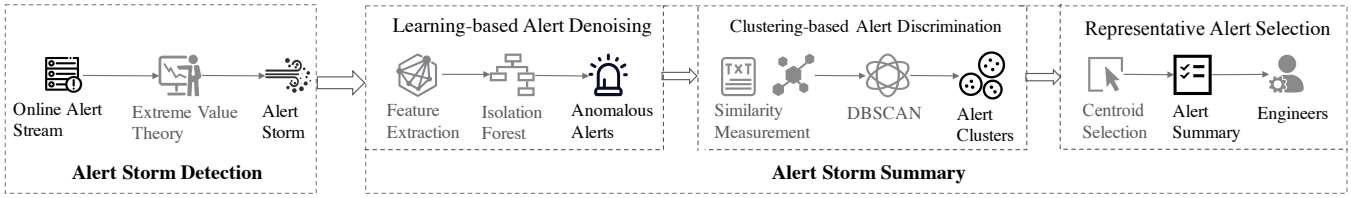


Figure 5: Overview of our approach

Figure 6: Overview of alert storm summary

ability to adapt to the evolution of the number of alerts and adjust the threshold dynamically, so as to tackle the highly dynamic online service systems. Figure 7 presents an alert storm case which can successfully detected by EVT. The #Alerts/minute exceeding the EVT-threshold (labeled in orange) are identified as alert storm, which are denoted in red points.

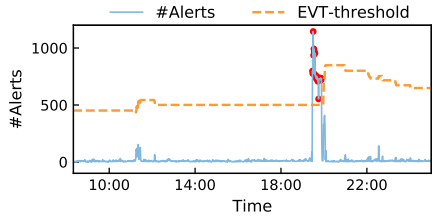


Figure 7: Illustration of alert storm detection using EVT

3.3 Alert Storm Summary

Once an alert storm case is detected, engineers will take actions to handle the alert storm and diagnose the problem. However, faced with such an overwhelming number of alerts, it is time-consuming and tedious for engineers to manually examine each alert for failure diagnosis. To tackle this challenge, we propose an alert storm summary method to extract a small set of representative alerts from the numerous alerts to assist engineers in handling alert storm. The method shown in Figure 6 includes three steps, i.e., learning-based alert denoising, clustering-based alert discrimination, and representative alert selection.

3.3.1 Alert Preprocessing. Before introducing detailed alert summary approach, we first need to preprocess the alert data for further analysis. Suppose we detect a case of alert storm at time t, we collect the alerts occurred during the time range $[t - w : t]$ as our input data and w is set as 5 minutes in our experiments, because there may be a time gap between the root cause alert and the alert storm due to the delay of alert generation. Inspired by the preprocessing techniques used in existing work [11], we adopt the following two steps to preprocess alert data.

Token Normalization. As presented earlier, the alert content is semi-structured and has some variables such as IP address and port number. For the purpose of further analysis, we are only interested in the textual structural information in alerts [11]. Therefore, we replace the variables by another common strings (e.g., replace various IP addresses with "ipaddr"). Token normalization significantly reduces the alert space and eliminates a lot of noises in the data. It is now much easier to detect structural patterns within the normalized alerts and conduct the following algorithm.

Stop-words removal. Formally, the alert content can be regarded as a bag of words (tokens). Terms like "the", "in" and "is" also known as stop-words do not carry much specific information in the context of the alert content. All stop-words are removed from the set of tokens based on a list of known stop-words.

3.3.2 Learning-based Alert Denoising. As presented in Section 2.4, there are some irrelevant alerts to the failure in alert storm. An intuitive denoising solution is to filter the alerts whose numbers do not perform a change point (spike) when the storm occurs (Figure 4(b)). However, for a large variety of alerts, monitoring the number of each alert and detecting spikes suffer from extremely high cost, which leads to its unavailability in practice. In our approach, we novelly formulate the problem of alert denoising as *anomaly detection*. We regard the regular alerts happening in normal states of the system as training set and the alerts happening during the alert storm as testing set. In this way, regular alerts are normal samples and the goal is to identify the anomalous alerts (different from regular alerts) in testing set.

A rich body of literature has been devoted to anomaly detection, e.g., One-Class SVM [1], Local Outlier Factor [2] and clustering-based methods [3]. However, these algorithms suffer from either high computational cost or poor performance. Isolation Forest (iForest) is a popular anomaly detection algorithm and has shown good performance with a linear time complexity [13], and thus we leverage iForest to detect anomalous alerts in our scenario.

Before applying iForest, we first extract several features from alert data. Based on our observations, anomalous alerts usually have some attributes which are very different from those of normal alerts. For example, an alert usually happens at night, but it happens in the morning. As shown in Table 1, the alert data has multi-attributes and we mainly focus on several important and indicative attributes. Combining with domain knowledge, eight features are extracted in our approach, including normalized alert content, server, service, type, alert time (hour, day of week, weekend or not) and frequency.

IForest [13] is a popular outlier detection algorithm and has been applied in various anomaly detection tasks. Technical details about iForest can be found in the existing work [13]. Given several features extracted from the above step, iForest can output an anomaly score for each alert. The larger the score is, the more likely the alert is abnormal. Then, we decide the ratio of anomalous alerts, i.e., how many alerts need to be retained and how many alerts need to be filtered. Specifically, we choose the number of anomalies (n_{anomaly}) based on the number of alerts in the training set (normal states),

which can be computed as $n_{\text{anomaly}} = n_{\text{test}} - \frac{\text{time span}_{\text{test}}}{\text{time span}_{\text{train}}} \times n_{\text{train}}$. In this way, the alerts with top-n_{anomaly} largest anomaly scores are retained and other alerts are filtered.

3.3.3 Clustering-based Alert Discrimination. After alert denoising, the majority of remaining alerts are related to the failure. As presented in Section 2.4, complex correlation exists in alert storm, including textual and topological correlation. To ensure the variety of our recommended alerts and reflect the failure from diverse aspects, we leverage the clustering technique to gather correlated alerts together, so that engineers only need to focus on a small set of alerts from different clusters, instead of all alerts. Alert clustering in our approach contains the following two steps.

Similarity Matrix Construction. Similarity measurement is a key component in clustering task. Given N alerts that need to be clustered, we create an $N \times N$ distance matrix to record the distance between two alerts. Given that there exist textual correlation and topological correlation in alert storm, we calculate the similarity from these two aspects, text and topology.

1) Textual similarity. Textual similarity is designed for detailed alert content. Here, we adopt Jaccard distance as metric to measure the textual similarity between two alerts [11]. The content of each alert (after data preprocessing) can be represented as a bag of words. Then, Jaccard distance between alerts a and b is defined as below:

$$Jaccard(a, b) = 1 - \frac{|bow(a) \cap bow(b)|}{|bow(a) \cup bow(b)|} \quad (1)$$

where $bow(a)$ means the bag of words of alert a. Jaccard distance is a number between 0 and 1, and is not sensitive to the position of each word in alert content.

2) Topological similarity. There are two kinds of topologies in real-world online service systems, i.e., software topology (service) and hardware topology (server). In practice, the topological relationships among all services and all servers can be represented a directed graph respectively, which can be usually obtained by Configuration Management Database (CMDB). The topological distance of two services/servers can be computed by the shortest path length between two nodes on the service/server topological graph. Thus the topological distance of two alerts can be computed as:

$$\text{topological}(a, b) = \text{path}_{\text{service}}(a, b) + \text{path}_{\text{server}}(a, b) \quad (2)$$

Finally, by normalizing the distance obtained by the above two steps to the interval of $[0, 1]$, we can compute the final similarity between two alerts (a and b) as follows:

$$\text{similarity}(a, b) = \alpha \times \text{textual}(a, b) + (1 - \alpha) \times \text{topological}(a, b) \quad (3)$$

The value of distance weight α is set to 0.6 based on the performance on the small part of our dataset, which will be discussed in detail in Section 4.4.2.

Clustering. Given the distance matrix that summarizes how the alerts relate to each other, various clustering methods can be applied, but different methods have different assumptions and advantages. Popular clustering algorithms contain partitional methods like k-means [14], density-based methods like DBSCAN [8], and hierarchical methods [16]. Based on experimental results, we choose DBSCAN due to the following reasons. First, we have no prior knowledge about the number of clusters (k), while DBSCAN can infer k based on the data. Second, it can discover clusters of arbitrary shape and can work with most distance measures [16]. Beside, it does not need to iteratively compute an explicit "centroid" and re-cluster at every iteration [7, 24]. DBSCAN requires two parameters: ϵ which is the density threshold, and *minPts* which is the number of minimum points to form a cluster. DBSCAN finds dense regions separated by low-density areas to form clusters. A cluster is expanded if its neighbors are dense, i.e., others that are similar to its core will be absorbed into it. We set *minPts* equals to 2 and set the value of ϵ using elbow method as suggested in [8].

3.3.4 Representative Alert Selection. After clustering-based alert discrimination, we can get k clusters and alerts in the same cluster are closely correlated to each other. Therefore, for each cluster, engineers only need to examine one representative alert that can capture the overall pattern of this cluster. In our approach, we select the representative alert by choosing the centroid of each cluster [12]. The centroid of a cluster is defined as the object which has the minimal average distance to other objects in the same cluster and can be calculated using the following equation:

$$\text{centroid} = \arg\min_{i \in \text{cluster}} \frac{1}{n} \sum_{j=1}^{n} \text{similarity}(i, j) \quad (4)$$

where n is the number of alerts in this cluster and j is the alert in this cluster. Finally, the selected representative alerts from each cluster are recommended to engineers for manual examination to diagnose failure. If engineers are interested in an alert, they can also examine other alerts in this cluster in depth.

4 EVALUATION

To evaluate the effectiveness of our proposed approach, we conducted an experimental study aiming to address the following research questions:

- **RQ5**: How accurate is the alert storm detection method?
- **RQ6**: How effective is the alert storm summary method for failure diagnosis?
- **RQ7**: How effective are components in alert storm summary?
- **RQ8**: How efficient is our alert storm summary method?

4.1 Dataset

As stated in Section 2, we obtained 3-year-long real-world alert data from a large commercial bank as our dataset, and collected 166 cases of alert storm as our experimental subjects from the historical failure tickets. The minimum and maximum number of alerts of these alert storm cases are 410 and 8739, respectively. Due to the long time span of our collected alerts, the patterns and characteristics of alerts may change largely over time. Therefore, we divided the 3-year-long data into three one-year-long datasets, which is also helpful to evaluate the generality of our approach to some degree. Table 2 shows the details of our datasets. We can observe the number of alerts increases over the years due to the deployment of some new services.

Table 2: Details of the experimental datasets

Datasets	Time span	#Alerts	#Storm cases
A	2016/07/01 ~ 2017/06/30	907851	49
B	2017/07/01 ~ 2018/06/30	1073491	56
C	2018/07/01 ~ 2019/06/30	1256313	61

4.2 Accuracy of Alert Storm Detection

The widely-used alert storm detection method in practice is based on a fixed threshold determined by domain experts. In our approach, we propose a novel method to identify alert storm accurately and adaptively by adopting EVT, a classical statistical theory. To evaluate the effectiveness of our EVT-based method, we compared the detection *precision*, *recall* and *F1-score* between our EVT-based method and the existing fixed threshold method (i.e., #alerts/minute > 500). Notice that we regard the continuous points exceeding the threshold as an alert storm case as shown in Figure 7. We obtained the ground truth of alert storm from historical failure tickets.

The results are shown in Table 3, and we found all the metrics achieved by our EVT-based method are larger than 0.9, demonstrating the great effectiveness of our alert storm detection method. Also, our EVT-based method outperforms the existing fixed threshold method in terms of all the metrics. Furthermore, according to the results from dataset A to dataset C, the fixed threshold method performs worse with the time increasing. The reason is that the system scale becomes larger (including deploying some new services), which causes many emerging alerts (shown in Table 2). The fixed threshold method cannot fit such a dynamic scenario, leading to poor performance. Besides, the fixed threshold method does not consider the context and is easy to generate false positives, which bring meaningless trouble to engineers. However, our EVT-based method can achieve stably good effectiveness on all the three datasets, showing the adaptability of our method. In summary, our alert storm detection method is indeed effective and adaptive.

Table 3: Performance comparison between our approach and fixed threshold method for detecting alert storm

Datasets	A			B			C		
Methods	P	R	F1	P	R	F1	P	R	F1
EVT	**0.92**	**0.96**	**0.94**	**0.90**	**0.97**	**0.93**	**0.95**	**0.96**	**0.95**
Threshold	0.82	0.99	0.90	0.75	0.92	0.83	0.59	0.91	0.72

4.3 Effectiveness of Alert Storm Summary

We evaluated the effectiveness of our alert storm summary method from the following two aspects:

- How much effort alert storm summary can reduce?
- How accurate are the alerts recommended by alert storm summary in identifying failures?

4.3.1 Effort Reduction. As mentioned earlier, a service failure tends to be along with the alert storm, so that engineers need to examine a large number of alerts for efficient troubleshooting. Here, we used $\frac{n_{\text{storm}}-n_{\text{examine}}}{n_{\text{storm}}}$ as the metric to measure the effort reduction, where n_{storm} is the number of input alerts which occurred during the storm and a few minutes before the storm, and n_{examine} is the number of alerts that need to be examined by engineers. As discussed in Section 1, it is time-consuming and tedious for engineers to examine each alert one by one in face of the alert storm. In practice, engineers may adopt some tricks to examine the numerous alerts. For example, engineers tend to first examine alerts with high severities (i.e., 1-critical in our datasets).

To demonstrate the ability of alert storm summary to reduce engineers' effort, we compared it with two baselines, i.e., severity trick and raw manual checking, whose results are shown in Table 4. "Denoising" and "Summary" denote the effort reduction achieved by alert denoising and the whole process of our alert storm summary, respectively. We observed that with our alert storm summary, the number of alerts need to be examined is reduced by more than 98% and engineers only need to investigate about 2% alerts to capture the detailed information about this alert storm from diverse aspects. In terms of raw manual checking without any tricks, engineers have to examine each alert one by one to identify the failure, which is tremendously labor intensive, slow and unreliable. Besides, critical alerts (severity=1) in our datasets account for more than 10%, indicating that the common-used severity trick used to handle alert storm only saves less than 90% effort, which is much smaller than that by our method (larger than 98%).

In summary, our alert storm summary is indeed effective in reducing the number of alerts need to be investigated, so as to save much effort for engineers.

Table 4: Comparison of effort reduction between our alert summary approach and compared methods

Datasets	Raw	Severity	Denoising	**Summary**
A	0%	88.7%	6.9%	**98.8%**
B	0%	85.6%	5.1%	**98.2%**
C	0%	84.1%	8.4%	**99.1%**

4.3.2 Accuracy. We further measured the accuracy of alert storm summary for failure diagnosis. Following the accuracy evaluation of log-based problem identification in [12], we counted the number of true positives (the number of examined alerts that are helpful to troubleshooting) and false positives (the number of examined alerts that are unhelpful to troubleshooting and engineers do not need to examine), which are acquired via manual analysis by engineers. We adopt the value of precision as evaluation metric which can be computed with $\frac{TP}{TP+FP}$.

Due to the limited number of available engineers and the limited time, they helped us identify which alerts are helpful to troubleshooting in dataset C only. It is reasonable since dataset C contains the most recent alerts and thus they can accurately identify them with the largest confidence. Table 5 shows the average precision results on dataset C achieved by two baselines, our approach, and our approach without the component of alert denoising (denoted as W/o denoising in this table). In general, our approach achieves much higher precision than the two baselines. Also, we observed that it is essential to adopt our alert denoising method, which can filter some irrelevant alerts to the failure so that the precision can be improved. However, our denoising method cannot remove all irrelevant alerts accurately, and there may exist some correlation between alerts to some degree. Therefore, there are still a small part of irrelevant and correlated alerts in the final recommended alerts, resulting in not very high precision of our approach. About the two compared methods, raw manual checking without any tricks needs to examine each alert, but there are many irrelevant and correlated alerts (Section 2.4) and engineers waste much effort on meaningless work, which leads to low precision. In terms of the severity trick, the severities of alerts are decided by manual rules without considering the correlation among alerts, which are also inaccurate to some degree.

In summary, our alert storm summary approach achieves the best precision and can minimize the meaningless effort for engineers.

Table 5: The precision comparison between our approach and compared methods

Method	Raw	Severity	W/o denoising	**Summary**
Precision	0.08	0.42	0.64	**0.75**

Finally, to further demonstrate the effectiveness of alert storm summary for troubleshooting, we take the alert storm case introduced in Section 2.4 (occurred at 2018/4/7 19:42) as example. Using traditional manual checking, engineers needed to examine more than 495 alerts (occurred during 19:37 ~ 19:42) and spend 48 minutes mitigating the failure. However, using our alert storm summary approach, 11 representative alerts are recommended as shown in Figure 8. Due to the limit of space, we only present the alert content and ignore other attributes. The alerts labeled in blue color imply the root cause (database server down) to some degree. In this way, engineers only need to investigate several valuable alerts and spend several minutes mitigating the failure, instead of examining hundreds of redundant alerts. This comparison demonstrates that our alert storm summary is indeed effective for failure diagnosis.

0. Oracle database connection detection alerts, TNS: no listener program
1. Weblogic JDBC pool status is not running. The alert item is NBANK: EntServer
2. Syslog alert, lan24 on system SDII1P1 has gone down due to lost connection with the link partner
3. Memory check timeout. Description/Type table : No response from remote host ipaddr
4. CPU usage: Can't get necessary data
5. Disk load current value: 100.00% exceeding configured threshold 95.00%
6. Oracle log alerts. ORA-01034: Oracle not available
7. Database transfer status is abnormal. Alert item is NBANKDB_STB
8. Node ping alert. Host check timed out after 10 seconds
9. Source ipaddr ping destination ipaddr, response timeout. Average RTT is 106ms, threshold is 100ms
10. System VCS cluster alert. SDII13P1_vcs has faulted in cluster CL_NBANK_DBSRAC_SHD

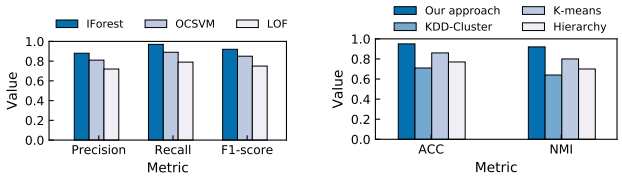
Figure 8: The summary result of the alert storm case introduced in Figure 3

4.4 Effectiveness of Components in Alert Storm Summary

We further investigated the performance of two key components in alert storm summary, i.e., learning-based alert denoising and clustering-based alert discrimination. Here, we also conducted the experiment only on dataset C as explained above.

4.4.1 Learning-based Alert Denoising. The goal of alert denoising is to filter some alerts that are irrelevant to the alert storm. As discussed in Section 3.3.2, we formulate the alert denoising problem as anomaly detection. Naturally, we adopt precision and recall as metrics, which can be computed as $\frac{\#TP}{n_{\text{anomaly}}}$ and $\frac{\#TP}{\#\text{truth}}$, respectively, where $\#TP$ is number of the detected anomalous alerts that are true anomalies, n_{anomaly} is the number of anomalous alerts given by the algorithm, which has been introduced in Section 3.3.2, and #truth is the number of true anomalous alerts labeled by engineers.

To demonstrate the effectiveness of the alert denoising, we replaced iForest in our approach with another two popular outlier detection algorithms, i.e., One-class SVM (OCSVM) [1] and Local Outlier Factor (LOF) [2]. Figure 9(a) presents the precision, recall, and F1-score comparison between iForest and the two baselines. We found that our approach identifies anomalous alerts that related to alert storm more accurately than OCSVM and LOF, achieving the F1-score larger than 0.9. Besides, we also took the efficiency into consideration. Based on our observations, for 1000 alerts, iForest, OCSVM and LOF required 3.1 seconds, 10.4 minutes and 32.9 seconds running time, respectively. Therefore, iForest with linear time complexity requires negligible running time, while OCSVM suffers from extremely high computational complexity, which leads to unavailability in practice.



(a) Alert denoising (b) Alert clustering

Figure 9: The effectiveness of two key components in alert storm summary

4.4.2 Clustering-based Alert Discrimination. To evaluate the performance of alert clustering, we adopted two popular clustering metrics, i.e., accuracy and Normalized Mutual Information (NMI) [15]. Accuracy discovers the one-to-one relationship between resulting clusters and true classes, and measures the extent to which each cluster contains alerts from the corresponding class. NMI is an information-theoretic measure based on the mutual information of the true classes and the resulting cluster, normalized using the entropy of each. Note that ACC and NMI lie in the range of 0 to 1, where one is the perfect clustering result and zero is the worst.

Clustering performance. Alert clustering is a vital component in our approach, but it is not first proposed in this paper. Lin et al. proposed an alert clustering approach (KDD-Cluster) using Jaccard distance and graph-cut clustering algorithm in KDD'14. However, the goal is to extract information automatically and gain some insights from a novel visualization of the clustering results, not designing for handling alert storm [11]. Figure 9(b) shows the average accuracy and NMI comparison between our approach and KDD-Cluster. We observed that our approach performs better than KDD-Cluster, since the latter only considers textual similarity between alerts and graph-cut performs not very well in our scenario.

Furthermore, to demonstrate the effectiveness of our used clustering algorithm DBSCAN [8], we adopted another two popular clustering algorithms, i.e., k-means [14] and hierarchy clustering [16], to compare with DBSCAN. Please note that we chose the number of clusters (k) for these two baselines based on Silhouette Analysis [18]. The parameters of DBSCAN (ϵ and minPts) have been introduced in Section 3.3.3. The accuracy and NMI comparison are presented in Figure 9(b), which shows that DBSCAN adopted in our approach is indeed more effective compared with the other two clustering algorithms. Besides, DBSCAN has the ability to discover the best k automatically. However, k-means and some other algorithms need to run many times under different k, and then choose the best k, which suffer from high computational cost.

The impact of distance weight. As introduced in Section 3.3.3, alert clustering in our approach has a parameter, i.e., distance weight (α in Eq.(3)). To study the impact of the distance weight on clustering performance, we varied the value of α from 0 to 1 and Figure 10 shows the accuracy and NMI values achieved by different α. Clearly, the alert clustering method achieves the best performance when the value of α is equal to 0.6. In our study, we chose the best α based on

a small sample of dataset. Besides, the results also demonstrate that both textual distance and topological distance play key roles in our approach. Both only using textual distance ($\alpha = 1$) and only using topological distance ($\alpha = 0$) perform worse than the combination of two types of distance.

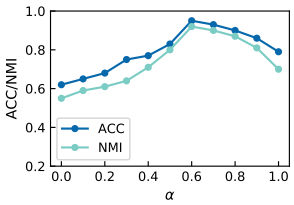


Figure 10: The impact of distance weight

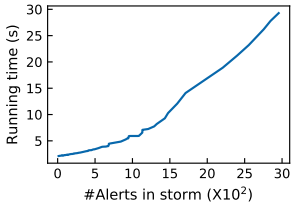


Figure 11: The efficiency of alert summary approach

4.5 Efficiency of Alert Storm Summary

Considering our alert storm summary approach is used to handle online alert storm and assist troubleshooting in practice, short response time is desired. Otherwise, service quality and user experience will be destroyed if engineers wait for a long time to get the summary results. Figure 11 presents the running time on some alert storm cases in our experimental datasets. We implemented our approach with Python and ran the program on a Dell PowerEdge R420 server with an Intel Xeon E5-2420 CPU and a 24GB memory. We found that when the number of alerts in storm is not very large (e.g., less than 1000), the response time of alert storm summary is very short (less than 7 seconds). As the number of alerts increases, the response time also increases, but still in an acceptable range. The total computational complexity of our approach is about $O(N^2)$. In real deployment, we can further reduce the response time by multi-process and some other optimization techniques.

5 DISCUSSION

5.1 Success Story

Our proposed approach including alert storm detection and alert storm summary has been successfully applied in a large commercial bank, which is used to assist engineers to handle alert storm intelligently. Before deploying our approach, engineers were insensitive to alert storm and tired of so many alerts. The majority of engineers tended to ignore alert storm until the failure was exposed by user complaint. In this way, failures cannot be discovered and mitigated in time, which may lead to unsatisfactory user experience and huge economic loss. After integrating our approach, engineers are able to detect potential failures more accurately and further shorten the time to diagnose the failure. Taking a recent case as an example, a disk failure incurred a case of alert storm (722 alerts in total), which can be detected by EVT-based detection method successfully. Then equipped with the alert storm summary, engineers only examined 31 alerts and took about eight minutes to locate the root cause and mitigate the failure. Therefore, based on the feedbacks from many service teams, our approach indeed enables more efficient failure discovery and failure diagnosis in practice. Besides, they appreciated that our approach can reduce the key metric "Mean Time to Repair" (MTTR) significantly.

5.2 Lessons Learned

Better alert rules. In our study, we found that there exist lots of meaningless and redundant alerts caused by unreasonable alert rules. In real world, engineers spend much time setting alert rules for a large variety of monitoring data. However, due to system complexity, it is hard to set perfect rules, and unreasonable rules often lead to lots of false positives and false negatives in alert generation. These meaningless alerts would bring trouble to engineers and waste much time. Besides, different engineers may write different rules based on their personal maintenance experience and intuitions. Therefore, how to set reasonable alert rules with minimum manual efforts is a big challenge in practice.

More intelligent alert management system. Existing alert management systems in practive only contain some simple techniques to process alert data, such as alert de-duplication and alert assignment. However, it is far from satisfying in reality. Exploring some novel algorithms and deploying in alert management system are indeed desired. For example, how to design an accurate and adaptive alert ranking strategy (instead of rule-based) to accommodate to dynamic online service system, how to assign various alerts to corresponding responsible engineers automatically and accurately, how to predict failures in advance based on historical alerts, and how to reduce the number of alert in advance to avoid false alert storm. These issues can assist engineers handling alerts more intelligently and can be our future work.

5.3 Threats to Validity

We have identified the following threats to validity in our study:

Noises in labeling: Based on historical failure tickets, the experienced engineers manually labeled the ground truth carefully for our evaluation. Noises (false positives/negatives) may be introduced during the manual labeling process. However, given that the engineers are experienced experts with rich domain knowledge and the ground truth is provided based on historical tickets, we are confident that the amount of noises in labeling is small (if it exists).

Subject selection bias: In our work, we obtained a 3-year-long alert dataset from a large commercial bank through its unified alert management system and collected 166 alert storm cases for our study (Section 2 and 4). Since the alert management platform gathers all alerts from hundreds of online service systems of this bank, our dataset is typical, large-scale and contains various alerts. However, the number of alert storm cases is still limited. In the future, we will reduce this threat by evaluating our approach on more alert storm cases from different companies.

Evaluation metics and parameters setting: We evaluated our approach from multiple aspects and adopted suitable metrics for each research question. One limitation of our metrics is effort reduction (Section 4.3.1). It is because that engineers may not investigate all alerts in the storm actually. However, it is difficult to determine the best practice adopted in real world. In the future, we will reduce this threat by considering more metrics to more sufficiently measure them. In terms of the parameters setting, we adopt some default parameters provided by the original algorithm (e.g., DBSCAN). For the parameters need to be set additional (e.g., distance weight α), we set them based on the performance of a small sample of dataset (Section 4.4.2). In our future work, we will further explore the impact of various parameters.

6 RELATED WORK

6.1 Alert Management

Despite a great deal of efforts have been devoted into alert management, including alert aggregation [22], alert correlation [19], alert ranking [10] and alert clustering [11], investigation about alert storm and effective methods to handle alert storm remain elusive. Lin et al. [11] proposed to cluster semi-structured alert texts to gain some insights from the clustering results, so as to improve operational efficiency. However, it only consider the textual similarity among alerts, while there exists topological correlation in real world. Besides, as stated in Section 2.4, there are some regular alerts in alert storm, and thus it is necessary to filter these alerts. Therefore, their approach is limited to handle alert storm in our scenario, which has been demonstrated in Section 4.4.2.

6.2 Problem Identification

Recently, many existing works focus on problem identification and failure diagnosis to ensure the quality of service in software maintenance [9, 10, 12]. To our best knowledge, this is the first work that focuses on problem identification in alert storm. Jiang et al. [10] proposed to identify problems by ranking the importance of alerts and the top ranked alerts are more likely to be problem. This work compared the metric value in alert data (e.g., CPU utilization) to determine the importance of alerts. However, it is only applicable to KPI alerts with threshold and strongly assumes the linear relationship between two alerts. In practice, there exist various alerts (Table 1) and not all alerts are generated by threshold rules.

In the field of log analysis, some related works aim to identify problems from a large volume of log data. Lin et al. [12] proposed LogCluster to cluster log sequences and pick the center of each cluster to identify problem. Rosenberg et al. [17] proposed to add dimension reduction technique into LogCluster to solve the high-dimensional log sequence vector, so as to achieve a better performance. Different from them, our work focuses on alert data rather than log data, and log-based problem identification approaches cannot be applied directly to our problem due to the differences between alert data and log sequences, but the alert clustering technique in our approach is indeed inspired from LogCluster.

7 CONCLUSION

Alert storm is a frequent phenomenon in the maintenance of online service systems. Faced with such an overwhelming number of alerts, it is very time-consuming and tedious for engineers to manually examine each alert to diagnose the real failure. To better understand the alert storm in practice, we conduct the first empirical study to investigate alert storm based on the large-scale real-world alert data from a large commercial bank. Based on the insights gained from the empirical study, we propose a novel approach to handling alert storm, which can detect the alert storm accurately and recommend a small set of representative alerts to engineers from the numerous alerts, so as to save engineers' effort in diagnosing failure. We evaluate the effectiveness of our approach based on real-world dataset, and the results show that our approach is indeed effective in both alert storm detection and alert storm summary. Besides, real deployment in practice shows our approach enables more efficient failure discovery and failure diagnosis.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their valuable feedbacks. This work has been supported by the Beijing National Research Center for Information Science and Technology (BNRist) key projects and National Key R&D Program of China 2019YFB1802504.

REFERENCES

[1] Mennatallah Amer, Markus Goldstein, and et.al. 2013. Enhancing one-class support vector machines for unsupervised anomaly detection. In *SIGKDD*. ACM.
[2] Markus M. Breunig, Hans-Peter Kriegel, and et.al. 2000. LOF: Identifying Density-based Local Outliers. In *SIGMOD*. ACM.
[3] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 15.
[4] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, and Feng et al. Gao. 2019. An empirical investigation of incident triage for online service systems. In *ICSE: SEIP*. IEEE Press, 111–120.
[5] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Dang Yingnong, and Dongmei Zhang. 2019. Continuous Incident Triage for Large-Scale Online Service Systems. In *2019 34st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, to appear.
[6] Laurens De Haan and Ana Ferreira. 2007. *Extreme value theory: an introduction*. Springer Science & Business Media.
[7] Min Du and Feifei Li. 2017. ATOM: efficient tracking, monitoring, and orchestration of cloud resources. *IEEE TPDS* 28, 8 (2017), 2172–2189.
[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, Vol. 96. 226–231.
[9] Shilin He, Qingwei Lin, Jian-Guang Lou, and et al. 2018. Identifying impactful service system problems via log analysis. In *ESEC/FSE*. ACM, 60–70.
[10] Guofei Jiang, Haifeng Chen, Kenji Yoshihira, and Akhilesh Saxena. 2011. Ranking the importance of alerts for problem determination in large computer systems. *Cluster Computing* 14, 3 (2011), 213–227.
[11] Derek Lin, Rashmi Raghu, Vivek Ramamurthy, Jin Yu, and et al. 2014. Unveiling clusters of events for alert and incident management in large-scale enterprise it. In *SIGKDD*. ACM, 1630–1639.
[12] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, and et al. 2016. Log clustering based problem identification for online service systems. In *ICSE*. ACM, 102–111.
[13] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 413–422.
[14] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*.
[15] Christopher Manning, Prabhakar Raghavan, and et al. 2010. Introduction to information retrieval. *Natural Language Engineering* 16, 1 (2010), 100–103.
[16] Lior Rokach and Oded Maimon. 2005. Clustering methods. In *Data mining and knowledge discovery handbook*. Springer, 321–352.
[17] Carl Martin Rosenberg and Leon Moonen. 2018. Improving problem identification via automated log clustering using dimensionality reduction. In *ESEM*. ACM, 16.
[18] Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.
[19] Saeed Salah, Gabriel Maciá-Fernández, and JesúS E DíAz-Verdejo. 2013. A model-based survey of alert correlation techniques. *Computer Networks* 57, 5 (2013), 1289–1317.
[20] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and et al. 2017. Anomaly detection in streams with extreme value theory. In *SIGKDD*. ACM, 1067–1075.
[21] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, and et.al. 2018. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In *WWW*.
[22] Jingmin Xu, Yuan Wang, Pengfei Chen, and Ping Wang. 2017. Lightweight and Adaptive Service API Performance Monitoring in Highly Dynamic Cloud Environment. In *SCC*. IEEE, 35–43.
[23] Nengwen Zhao, Jing Zhu, Rong Liu, Dapeng Liu, Ming Zhang, and Dan Pei. 2019. Label-Less: A Semi-Automatic Labelling Tool for KPI Anomalies. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1882–1890.
[24] Nengwen Zhao, Jing Zhu, Yao Wang, Minghua Ma, Wenchi Zhang, and et.al. 2019. Automatic and Generic Periodicity Adaptation for KPI Anomaly Detection. *IEEE Transactions on Network and Service Management* (2019).
[25] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, and et al. 2018. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE TSE* (2018).
[26] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, and et al. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *ESEC/FSE*. ACM, 683–694.