Online Summarizing Alerts through Semantic and Behavior Information

Jia Chen
Fudan University
Shanghai, China
chenj17@fudan.edu.cn

Peng Wang*
Fudan University
Shanghai, China
pengwang5@fudan.edu.cn

Wei Wang
Fudan University
Shanghai, China
weiwang1@fudan.edu.cn

ABSTRACT

Alerts, which record details about system failures, are crucial data for monitoring a online service system. Due to the complex correlation between system components, a system failure usually triggers a large number of alerts, making the traditional manual handling of alerts insufficient. Thus, automatically summarizing alerts is a problem demanding prompt solution. This paper tackles this challenge through a novel approach based on supervised learning. The proposed approach, OAS (Online Alert Summarizing), first learns two types of information from alerts, semantic information and behavior information, respectively. Then, OAS adopts a specific deep learning model to aggregate semantic and behavior representations of alerts and thus determines the correlation between alerts. OAS is able to summarize the newly reported alert online. Extensive experiments, which are conducted on real alert datasets from two large commercial banks, demonstrate the efficiency and the effectiveness of OAS.

CCS CONCEPTS

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

KEYWORDS

alert summary, online service systems, system maintenance

ACM Reference Format:
Jia Chen, Peng Wang, and Wei Wang. 2022. Online Summarizing Alerts through Semantic and Behavior Information. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510003.3510055

1 INTRODUCTION

Online service systems, such as online banking and search engine, have become indispensable parts in our daily life. Organizations of all sizes struggle with a common problem in infrastructure and operational management. Thousands of automated alerts with semi-structured text are generated every day from hundreds of infrastructure tools [13].

As the scale and complexity of online service systems increase rapidly, failures are inevitable in practice. There are many factors that cause failures, such as network failures, hardware problems, software bugs, temperature, humidity, and even human errors. In addition to affecting customer satisfaction, failures can even cause huge economic losses, especially for online systems involving financial services, such as banking systems, online supermarkets, and stock trading systems. In 2019, the estimated cost of the one-hour downtime for Amazon on Prime Day, which is the biggest sale event of the year for Amazon, is up to 100 million dollars [3].

To detect the failure timely, maintenance engineers monitor various data of online service system in real time, such as KPIs [26, 34, 35], logs [5, 7, 9, 14, 18, 30], and traces [15, 20, 21, 36]. When the monitoring data is abnormal, corresponding alerts will be reported, which describes the phenomenon of the underlying failure, and then maintenance engineers will be notified immediately to handle the alerts and fix the failure. Thus, alerts are essential data for engineers to maintain the service system.

In a real scenario, a failure usually triggers a large number of alerts, due to the interaction between system components, resulting in a large number of alerts in a day, or even an alert storm due to the overwhelming number of alerts in a short period of time [31]. Although maintenance engineers commonly define correlation rules to summarize alerts of the same failure, thereby reducing the intensity of their maintenance work. However, manually defining correlation rules is a labor intensive and time-consuming process, and it is hard to cover all correlations between alerts. In consequence, designing approach to automatically summarize alerts into incidents is an urgent requirement.

There are a few works focusing on automatically summarizing alerts [13, 31]. Such works claim that since correlated alerts describe the same failure, their contents may be similar, and the propagation of a failure within the system follows the topology of system components, thus the topological relationship between system components may reveal the correlation between alerts of the failure. Therefore, existing works summarize alerts according to two factors, the Jaccard similarity between alert contents [13, 31] and the topological relationship between system components [31].

However, these approaches have following drawbacks. First, alerts triggered by one failure may have totally different contents. For example, as shown in Table 1, although these alerts are all caused by the failure "NTP start error", the first two alerts have no common words. In this case, the Jaccard similarity [31] cannot correlate these alerts successfully. Second, the topology of system components may be inaccurate and hard to utilize. In general, the topology is derived from the CMDB system (Configuration Management Database). However, the update of CMDB is a tedious and challenging work.

*Peng Wang is the corresponding author.

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9221-1/22/05…$15.00
https://doi.org/10.1145/3510003.3510055

Even worse, in a cloud-based system, there usually exist many service containers running on one server, and corresponding system topology changes frequently.

In addition to above issues, existing works are all unsupervised. However, many companies save a large number of work orders and failure reports, which are natural sources of labels for alert summarizing. Specifically, a maintenance engineer is usually required to write a report after fixing a failure. Such failure report records all fix actions performed by the engineer, the root cause of the failure, as well as alerts triggered by the failure. These data contain a wealth of valuable expert experience and knowledge. Therefore, instead of blindly extracting the correlation from original alerts, a more robust choice is to learn the alert correlation from such reports in a supervised fashion.

In this paper, we propose a novel supervised framework, OAS (Online Alert Summarizing), to summarize alerts. OAS leverages two types of alert information, semantic information and behavior information. Two deep learning models, ASR and ABR, are proposed to extract these information respectively. ASR (Alert Semantics Representation) extracts the semantics of alerts, which aggregates the contextual information of alert words according their importance. Meanwhile, ABR (Alert Behavior Representation) mines the common behavior pattern between alerts from the alert occurrence series. Then, to deal with the complexity of the alert correlation, instead of determining the correlation by simply setting a fixed threshold, we design a deep learning model, ACT (Alert CorrelaTion), to combine above two types of alert information and determine the correlation between alerts automatically.

In summary, the core research problem in this paper is how to automatically summarize alerts of a service system online. We adopt Design Science Research [25] as our research methodology. We investigate the pros and cons of existing approaches and propose a new solution, OAS, to solve the research problem. Furthermore, a prototype of OAS is implemented and we validate it on real alert datasets from two large commercial banks. The contributions of this paper are as follows:

- We propose OAS (Online Alert Summarizing), to automatically summarize alerts online, which contains four main components, ASR, ABR, ACT and online summarizing. ASR integrates the contextual information of alert words according to their importance. ABR is able to capture the underlying common behavior information between correlated alerts. ACT utilizes the strengths of ASR and ABR to determine the correlation between alerts. Online summarizing adopts these trained models to summarize the newly reported alert online by a time window.
- We evaluate our approaches with alerts from two large commercial banks, A and B, and compare our approaches with the state of the art. Experimental results verify the efficiency and effectiveness of our approach.
- We share the prototype of our approach, and an alert dataset of a large commercial bank, B, in which the confidential information and some sensitive alert types are removed. Since the alert is usually confidential for a company, there is few public alert datasets of a service system yet.

Significance. The significance of this paper is as follows. First, to the best of our knowledge, this is the first supervised approach of alert summarizing that tries to learn knowledge from failure reports. Second, this is the first time to directly mine the behavior information of alerts from the alert occurrence series. Third, we design three deep learning approaches, ASR, ABR and ACT to automatically summarize alerts, and experimental results show that our approaches can achieve the best effectiveness.

The rest of this paper is organized as follows. We first present the related work in Section 2. The motivation of our study is summarized in Section 3. Then, in Section 4, we give the preliminary of our approaches. Section 5~7 present details of our approaches. We demonstrate the effectiveness and efficiency of our approaches by experiments based on real dataset in Section 8. In Section 9, we propose lessons we learned from this study. Finally, in Section 10, we conclude our work and discuss its future extension.

2 RELATED WORK

In recent years, as alerts become the main data for maintenance engineers to detect and analyze system failures, tremendous efforts have been devoted into alert analysis in both academia and industry. Some works focus on alert summarizing [13, 27, 31], some focus on alert prioritization [11, 33], and some focus on incident prediction [4, 32].

For alert summarizing, Lin et al. [13] try to correlate alerts by alert contents to gain some insights of the system failure, and thus improve the efficiency of maintenance engineers. Since alert storm, which brings an overwhelming number of alerts, is a great challenge for maintenance engineers, Zhao et al. [31] propose AlertStorm to combine alert contents and system topology to correlate alerts of the same failure. Both of them adopt Jaccard to measure the textual relationship between alerts, which only considers the number of common words of alerts and fails to capture the hidden semantics of alerts. Xu et al. [27] exclusively recognize API performance problems in a Cloud platform and correlate API alerts by the dependency of APIs. However, the system topology and API information are usually inaccurate and hard to utilize, because maintaining these information is a tedious and challenging work.

In addition to above approaches that are specific for alerts, there are some generic event summarizing approaches [10, 12, 28]. Based on MDL (Minimum Description Length) principle [24], such approaches try to find a set of frequent patterns which can represent the system behavior model and well summarize the event sequence. SeqKrimp [12] is a two-step approach. It first generates a set of candidate frequent patterns and then greedily selects a set of patterns that minimize the description length of the event sequence. GoKrimp [12] directly summarizes the event sequence by patterns composed of frequent event types. CSC [10] is similar to GoKrimp, but it supports the overlap between different pattern occurrences. SWIFT [28] is an online approach to summarize the event stream by mining frequent patterns with a sliding window. Since many alert types may have quite low frequencies in real-world situations, such approaches based on frequent patterns are not suitable for summarizing alerts. In this paper, experiments conducted on real alert datasets demonstrate that these approaches are not applicable to real scenarios.

For alert prioritization, Jiang et al. [11] propose a peer-review mechanism to rank the importance of alerts. Their work helps maintenance engineers to determine which alert should be analyzed first in the problem determination process. Zhao et al. [33] present AlertRank, an automatic and adaptive framework for identifying severe alerts. Specifically, AlertRank first extracts a set of interpretable features, such as alert content and alert frequency, and then adopts XGBoost ranking algorithm to identify severe alerts out of all incoming alerts. According to experiments, AlertRank significantly save troubleshooting time on non-severe alerts for maintenance engineers.

For incident prediction, Chen et al. propose AirAlert [4] to predict the occurrence of outages and diagnose the root cause after they indeed occur. AirAlert analyzes relationships between outages and alerting signals by leveraging Bayesian network and predict outages using a robust gradient boosting tree based classification approach. By forecasting outages before they actually happen and diagnosing their root cause, AirAlert is able to minimize service downtime and ensure high system availability. Zhao et al. [32] propose eWarn to online forecast whether an incident will happen in the near future based on alerts. eWarn first extracts a set of alert textual features and statistical features to represent omen alert patterns for an incident, then eWarn incorporates the multi-instance learning to reduce the influence of noisy alerts. Finally, eWarn builds a classification model via machine learning and generates an interpretable report for its prediction. eWarn has been applied to two large commercial banks in practice, which proves its practicability and effectiveness.

3 MOTIVATION

In this section, we use an illustrative example to motivate our approach. Table 1 shows an alert snippet of a large commercial bank, A, in which the alerts, $e_1, e_2, \cdots, e_5$, are all caused by the same failure, "NTP start error". By removing variable parameters and stop words from contents, these alerts can be classified into three types, E_1, E_2, and E_3. Alerts of the same type have the same parsed content. In Table 1, e_1 and e_3 belong to E_1, e_2 and e_4 belong to E_2, and e_5 belongs to E_3.

Our study aims to automatically summarize such alerts into a group, named as *incident*, thereby reducing the number of alerts analyzed by maintenance engineers. To mine the correlation between alerts, in this paper, we leverage two types of alert information, semantic information and behavior information.

3.1 Semantic Information

As shown in Table 1, contents of the alert of E_2 and E_3 have some common key words, such as "NTP" and "start", which reveals the common semantic information between alerts. However, mining such common semantic information is not trivial, because it is often that correlated alerts have only a few common words and most words in their contents are different. Therefore, popular approaches, like Jaccard [31] and Word2Vec [19], may fail to capture such faint common semantic information.

To attack such problem, in this paper, we propose a deep learning based model, named ASR (Alert Semantics Representation), to extract the semantic information of alerts. ASR not only mines the contextual information of each alert word, but also considers the contribution of each word to the overall alert semantics. As a result, the contextual information of alert words is integrated based on their semantic contribution. In the previous example, ASR will pay more attention to the words, "NTP" and "start", than the other words in the alert of E_2 and E_3.

3.2 Behavior Information

Compared to correlating alerts of E_2 and E_3, correlating those of E_1 is more challenging since there are no common words between E_1 and E_2 (or E_3). Thus, we leverage the behavior information lurking in co-occurrences of alerts. The rationale is that correlated alerts should occur together in higher probability. For example, Figure 1 shows the occurrence series for alerts of E_1 and E_2 in one day, which are formed by counting alert occurrences of each type per one minute in a day. It can be seen that 1) alerts of E_1 and E_2 always occur in the same time periods, $[t_1, t_2]$, $[t_3, t_4]$, and $[t_5, t_6]$, which correspond to three distinct failures; 2) the fluctuations of these two series are similar.

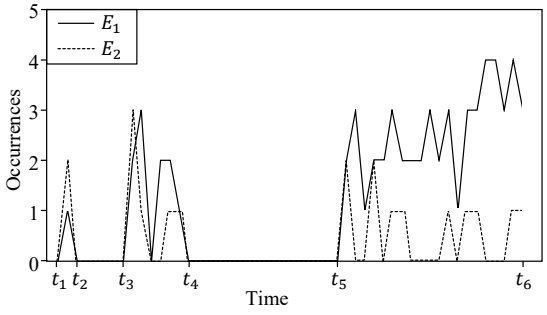


Figure 1: The occurrence series of correlated alerts.

The straightforward approach is to measure the similarity of two occurrence series using Euclidean Distance (ED) or Dynamic Time Warping (DTW). However, although the two occurrence series in Figure 1 are similar in $[t_1, t_2]$ and $[t_3, t_4]$, they have different shapes in $[t_5, t_6]$. This is because the occurrence of alerts is affected by many factors, like alert detection mechanism, failure severity, failure duration, and so on. Another alternative is to mine frequent patterns (or sequences) [10, 12, 28]. But they can only find incidents with frequent alert types. The infrequent ones, although more important, will be missed.

In this paper, we propose a data driven model, ABR (Alert Behavior Representation), to represent the behavior information of an alert. Inspired by Skip-Gram [19], ABR capture the commonality between the occurrence series of correlated alerts. Moreover, ABR utilizes the supervised learning to leverage the expert knowledge. The advantage is that even the occurrence series of correlated alerts is not intuitively similar, ABR is still able to capture their common behavior information.

3.3 Combining Semantic and Behavior Information

After extracting the semantic information and the behavior information of the alert, it is challenging to effectively combine them

Table 1: A snippet of an alert sequence

No.	ID	Timestamp	Content
e_1	E_1	2018/5/1 18:00	There isn't effective configuration in /var/opt/conf/bank_name_check.conf, please retouch the file.
e_2	E_2	2018/5/1 18:01	Running NTP start-up check script failed, tried 1 time(s).
e_3	E_1	2018/5/1 18:01	There isn't effective configuration in /var/opt/conf/bank_name_check.conf, please retouch the file.
e_4	E_2	2018/5/1 18:02	Running NTP start-up check script failed, tried 2 time(s).
e_5	E_3	2018/5/1 18:03	The NTP daemon server has not been start correctly.

to finally determine the correlation between alerts. The naive approach is to consider semantic information and behavior information separately. Given two alerts, computing the similarity between their semantic information or behavior information, as long as one similarity exceeds the threshold, these two alerts are considered as correlated. However, due to the complexity and variety of the alert mechanism, it is infeasible to manually set the appropriate threshold for alerts of all possible failures.

We propose a neural network, called ACT (Alert CorrelaTion), to combine the two types of alert information from ASR and ABR, and determine the correlation between alerts. In the training stage, we generate a set of alert pairs, each of which has a label, correlated or uncorrelated. The ACT network automatically learns the optimal combination mechanism from the labelled data.

3.4 Supervised vs. Unsupervised Approach

Unlike the state of arts [13, 31], we utilize supervised learning approaches to summarize alerts. We make this choice due to following reasons. First, in many companies, the labelled data is easy to obtain. Failure reports are natural labelled data [31]. Each failure report includes related alerts, the failure information, and so on. Therefore, any two alerts belonging to one failure report can be considered as correlated. Second, incident recognition is a subjective task. It may happen that different experts have opposite opinions about the correlation between alerts, which is influenced by the maintenance system and mechanism. Thus, the supervised approach is more appealing to maintenance engineers.

4 BACKGROUND AND APPROACH OVERVIEW

In this section, we present the necessary background knowledge and give the approach overview.

4.1 Alert Preprocessing

Given an alert sequence, we first preprocess alerts in four steps. In the first step, we remove variables in the alert content. There are numerous works on parsing alerts [1, 8, 29]. As Drain [8] is a popular online parser, we adopt it as the parser of OAS. In the second step, we further remove stop words from the alert content, since stop words, such as "the", "a", and "and", do not carry much specific semantic information. In the third step, according to alert contents, we group alerts into different types, and thus alerts of the same type have the same alert content. In the last step, for an alert, we form its occurrence series by counting the number of alerts of the same type per α minutes in the past β minutes ($\alpha < \beta$). As a result, the alert content contains the semantic information of the alert, and the occurrence series contains the behavior information of the alert.

For convenience, we define the parsed alert sequence as $S = [e_1, e_2, \cdots, e_n]$. For an alert, e_i ($1 \le i \le n$), the timestamp is denoted as t_i. We have a set of alert types, $\{E_1, E_2, \cdots, E_m\}$, and each alert belong to one alert type. W_i ($|W_i| = l_i$) records the words in the content of e_i. The occurrence series of e_i is referred to as $F_i \in \mathbb{R}^{\lceil \frac{\beta}{\alpha} \rceil}$.

4.2 Overview

The objective of OAS is to utilize the semantic and behavior information of alerts to summarize alerts online. As a result, alerts are grouped into different incidents by OAS, and each incident contains alerts of the same system failure. In addition to reduce the number of alerts, compared to a single alert that only focuses on a local phenomenon of a failure, an incident can reflect the whole impact of the failure, thereby helping maintenance engineers efficiently locate and fix the failure.

Figure 2 shows an overview of OAS. OAS contains four main components, alert semantics representation (ASR), alert behavior representation (ABR), alert correlation (ACT), and online summarizing. OAS has two stages, training stage and summarizing stage. In the training stage, OAS trains the alert representation models, ASR and ABR, and the alert correlation model, ACT, offline according to the history alert sequence. In the summarizing stage, based on the trained models, OAS summarizes the newly reported alert online by a time window.

4.2.1 Training Stage. As shown in the left part of Figure 2, alerts in training dataset are first parsed to get alert contents and alert occurrences series. For each e_i in the training dataset, we obtain other alerts in window $[t_i - w, t_i]$ that belong to the same failure with e_i. Suppose there exist c_i alerts correlated to e_i during $[t_i - w, t_i]$, which are denoted as $R_i = [e_{r_1^i}, e_{r_2^i}, \cdots, e_{r_{c_i}^i}]$, ($1 \le r_j^i \le n$, $1 \le j \le c_i$). Alerts correlated to e_i can be collected from history failure reports. It should be noted that R_i may not be a complete set that contains all the alerts correlated to e_i. In fact, even an experienced domain expert can not guarantee that in each of his failure reports, all alerts belonging to the reported failure are found. Our approaches try to learn general relationships between available correlated alerts, and apply them to the newly reported alert in online alert stream.

Then, we train the semantics representation model, ASR, and the behavior representation model, ABR. Since ASR and ABR have no dependency, they can be trained separately. Finally, we train the alert correlation model, ACT, to measure the correlation between

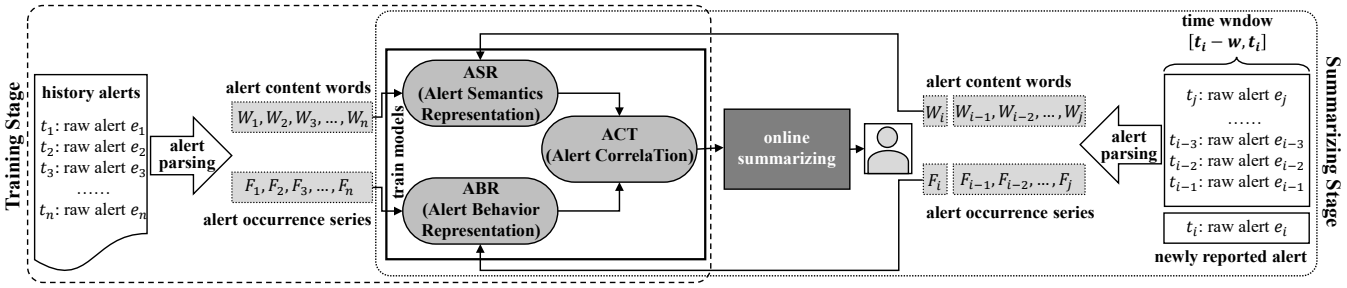


Figure 2: The architecture of OAS.

alerts by the mined semantic and behavior information from ASR and ABR.

4.2.2 Summarizing Stage. As shown in the right part of Figure 2, for the newly reported alert in the summarizing stage, e_i, after it is parsed, we can directly extract its semantic and behavior information by the representation models, ASR and ABR, which are trained in the training stage. Then, we measure the correlation between the newly generated alert, e_i, and previously reported alerts during a specified time window, $[t_i - w, t_i]$. According to alert correlations found by ACT, we propose a summarizing strategy to online group the newly reported alert, e_i, and its correlated alert into an incident.

5 ALERT REPRESENTATION

We propose two approaches, ASR (Alert Semantics Representation) and ABR (Alert Behavior Representation), to represent the semantic and behavior information of alerts respectively. As shown in Figure 2, extracting the semantic representation and the behavior representation are independent, thus there is no dependency between ASR and ABR.

5.1 Semantics Representation

Alerts belonging to the same system failure are likely to have similar semantic information. Although there are some existing approaches that can be used to represent the semantic information of alerts, such as Jaccard [13, 31], word embedding [7, 18], and topic distillation [33], they can not extract the complete semantics of the alert. The Jaccard-based approach naively correlates alerts with same words, ignoring the common semantics between different words. The word-embedding-based approach simply sums up embedding results of words in an alert, neglecting varying contributions of words to the overall semantics. The topic-distillation-based approach only distills the general topic of the alert, which are too crude to capture distinctive semantic details.

To represent the alert semantic information, we propose ASR (Alert Semantics Representation), which extracts the complete alert semantics based on the respective semantic contribution of each word in the alert content. For words in the alert content, ASR first mines the contextual information of each word, and then calculates the contribution of each word to the overall alert semantics. Finally, ASR aggregates the contextual information of alert words according to their contributions to the overall alert semantics. Therefore, the key issue is how to mine the contextual information of the alert word and how to calculate the contribution of the alert word to the overall alert semantics.

As shown in Figure 3, for each word in the alert content, ASR adopts a word embedding model, named CBOW (Continuous Bag-of-Words) [19], to mine the contextual information of the word, and utilizes IDF (Inverse Document Frequency) [14, 17] to calculate the semantic contribution of the word. More specifically, in CBOW, a neural network is trained to mine the contextual information of a word by extracting the common semantics between the word and its adjacent words. IDF is a widely used factor to measure the semantic importance of a word to the document, which is calculated by dividing the total number of documents by the number of documents that contain the target word. In ASR, the document is referred to the concatenate contents of correlated alerts in history alerts.

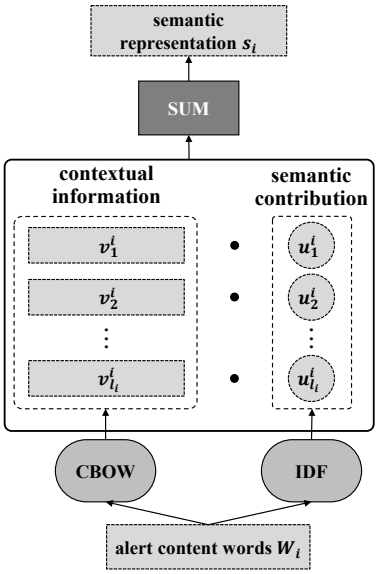


Figure 3: The process of representing the semantic information of the alert.

In the training stage, we separately train CBOW and IDF by contents in history alerts. Then, as shown in Figure 3, we can straightforwardly extract the semantic representation for an alert.

For each word in the alert, e_i ($1 \le i \le n$), we get the contextual information of the word by CBOW, denoted as v_j^i ($1 \le j \le l_i$), and the semantic contribution of the word by IDF, denoted as u_j^i. It should be note that in order to ensure $\sum_{j=1}^{l_i} u_j^i = 1$, u_j^i is normalized. At last, according to Equation (1), we can aggregate the semantics of words in the alert content to obtain the complete semantic representation of the alert, denoted as s_i.

$$s_i = \sum_{j=1}^{l_i} (v_j^i \cdot u_j^i) \tag{1}$$

5.2 Behavior Representation

In addition to semantics, alerts belonging to the same system failure usually also have common behavior information. Mining frequent patterns is a widely used approach to capture behavior correlations between alerts [10, 12, 28]. Because frequent patterns indicate co-occurrence correlations between alerts. In practice, however, some alert types have quite low frequencies to which frequent pattern mining approach is not applicable.

To represent the alert behavior information, we propose ABR (Alert Behavior Representation), which for the first time learns the commonality between alert occurrence series. ABR is inspired by the word embedding model, Skip-Gram [19]. Skip-Gram captures the common semantics between a target word and its adjacent words via a shallow neural network, which converts the encoding of the target word to the encoding of its adjacent words in a linear fashion. Similarly, as shown in Figure 4, ABR trains a shallow neural network to converts the occurrence series of a target alert to the occurrence series of its correlated alerts. As a result, the trained neural network can capture the underlying common behavior information between alerts.

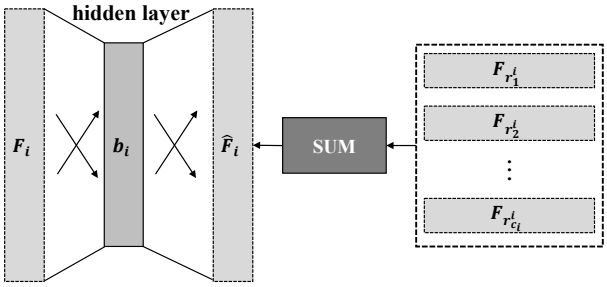


Figure 4: The process of representing the behavior information of the alert.

To train ABR in the training stage, for each history alert, e_i ($1 \le i \le n$), by Equation (2), we aggregate the occurrence series of alerts correlated to e_i, and denote the result as $\hat{F}_i$. Thus, since F_i contains the behavior information of e_i and $\hat{F}_i$ contains the behavior information of its correlated alerts, the commonality between F_i and $\hat{F}_i$ represents the same behavior information between e_i and its correlated alerts. Specifically, as shown in Figure 4, we try to transform F_i to $\hat{F}_i$ by a neural network, whose loss function is defined in Equation (3). In Equation (3), F_i' is the transformed result of F_i, Equation (3) measures the difference between F_i' and $\hat{F}_i$. After training, the result of the hidden layer in the neural network, denoted as b_i, is able to reserve the common behavior information between the occurrences series of e_i and its correlated alerts.

$$\hat{F}_i = \sum_{j=1}^{c_i} F_{r_j^i} \tag{2}$$

$$L_{ABR} = -\sum_{i=1}^{n} log(\frac{1}{1 + exp(-F_i' \cdot \hat{F}_i)}) \tag{3}$$

6 ALERT CORRELATION

With ASR and ABR, we can represent the semantic information and the behavior information of the alert, respectively. We thus propose a model, ACT (Alert CorrelaTion), to measure the correlation between alerts by aggregating the semantic and behavior representations of the alert. Before formally introducing ACT, let's consider a motivating question, for two alerts e_1 and e_2, when both their semantic and behavior representations are single-dimensional, how to determine if they are correlated. Generally, for the semantic representations of e_1 and e_2, the difference between them should be less than a threshold, and the same is true for the behavior representations. Formally, we define a simple auxiliary function, $f_{dis}(x, y, z)$, in Equation (4), which first calculates the square difference between x and y, and measure the difference between the square difference and z. Therefore, if e_1 and e_2 are correlated, both $f_{dis}(s_1, s_2, thrd_1)$ and $f_{dis}(b_1, b_2, thrd_2)$ are positive, where $thrd_1$ and $thrd_2$ are the thresholds for the semantic difference and the behavior difference, respectively. Identically, to ensure e_1 and e_2 are correlated, we can get $relu(f_{dis}(s_1, s_2, thrd_1)) \times relu(f_{dis}(b_1, b_2, thrd_2)) > 0$.

$$f_{dis}(x, y, z) = z - (x - y)^2 \tag{4}$$

ACT, shown in Figure 5, is a generalization of the above motivating example in a multi-dimensional situation. For two alerts, e_i and e_j ($1 \le i, j \le n$), inspired by the above example, ACT first calculates the square difference between each dimension of each type of alert representations. For the semantic representation, the result is denoted as $\bar{s}_{i,j}$, and for the behavior representation, the result is denoted as $\bar{b}_{i,j}$. In above example, $f_{dis}(s_1, s_2, thrd_1)$ and $f_{dis}(b_1, b_2, thrd_2)$ actually can be regarded as two linear transformation of square differences between each type of alert representations, where $thrd_1$ and $thrd_2$ are biases. Thus, ACT correspondingly performs linear transformation on $\bar{s}_{i,j}$ and $\bar{b}_{i,j}$, respectively. In addition, since $relu(f_{dis}(s_1, s_2, thrd_1)) \times relu(f_{dis}(b_1, b_2, thrd_2)) > 0$ when e_1 and e_2 are correlated, ACT similarly activates the transformation results by $relu$ function, and then aggregates the two types of alert representations by element-wise product. After further transformation and dimension reduction, ACT finally presents the correlation degree between e_i and e_j, $\hat{P}_{i,j} = [\hat{p}_1^{i,j}, \hat{p}_2^{i,j}]^\top$. $\hat{P}_{i,j}$ is a two-dimensional vector, in which $\hat{p}_1^{i,j}$ is the probability that the two alerts are correlated, and $\hat{p}_2^{i,j}$ is the probability that the two alerts are uncorrelated. The sum of $\hat{p}_1^{i,j}$ and $\hat{p}_2^{i,j}$ is 1, which is ensured by the $softmax$ function.

To train ACT in the training stage, for a history alert, e_i ($1 \le i \le n$), previous alerts during $[t_i - w, t_i]$ can be divided into two

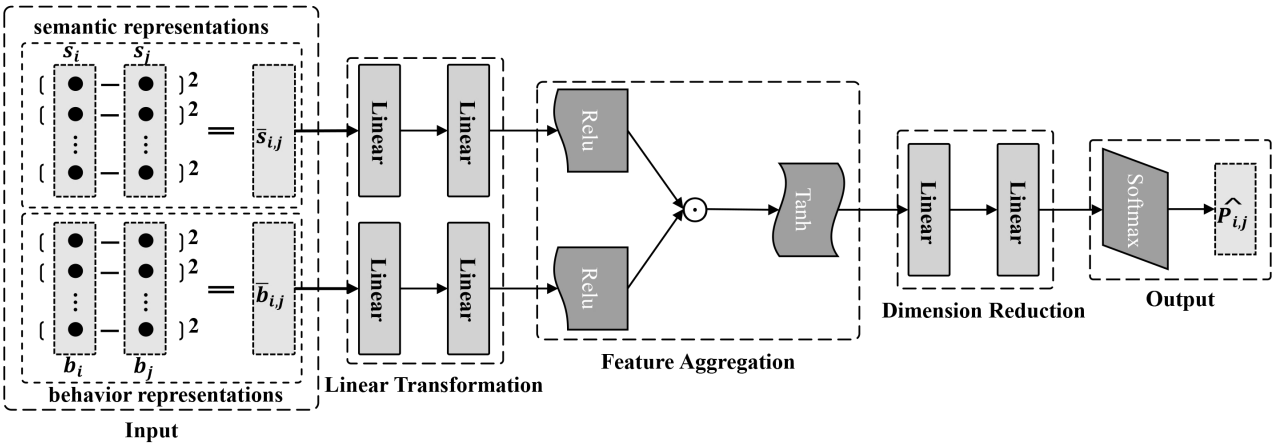


Figure 5: The structure of ACT.

groups, correlated alerts to e_i, denoted as R_i ($|R_i| = c_i$), and remaining uncorrelated alerts, denoted as H_i ($|H_i| = o_i$). Then, the real relationship between two alerts, e_i and e_j ($1 \le j \le n$), is formally defined as $P_{i,j} = [1, 0]^\top$ if e_i and e_j are correlated, otherwise $P_{i,j} = [0, 1]^\top$. Finally, we adopt Equation (5), which measures the difference between the determination of ACT and the ground truth, as the loss function of ACT.

$$L_{ACT} = \frac{1}{n}\sum_{i=1}^{n}\Big[\frac{1}{c_i}\sum_{e_j \in R_i}(\hat{P}_{i,j} - P_{i,j})^2 + \frac{1}{o_i}\sum_{e_j \in H_i}(\hat{P}_{i,j} - P_{i,j})^2\Big] \tag{5}$$

7 ONLINE SUMMARIZING

In the training stage, all models for alert representation and alert correlation are well trained offline. Therefore, in online summarizing stage, for the newly reported alert, e_i, and the previously reported alert, e_j, in the time window, $[t_i - w, t_i]$, we can easily represent their semantic information and behavior information by ASR and ABR, respectively. Then, according to ACT, we can obtain the correlation degree between the two alerts straightforwardly, which is defined as $\hat{P}_{i,j} = [\hat{p}_1^{i,j}, \hat{p}_2^{i,j}]$. Specifically, $\hat{p}_1^{i,j}$ indicates the probability that the alerts are correlated, and $\hat{p}_2^{i,j}$ indicates the probability that the alerts are uncorrelated. If $\hat{p}_1^{i,j} > \hat{p}_2^{i,j}$, e_i and e_j may belong to the same system failure. Therefore, we use the Equation (6) to find the alert most correlated to e_i during $[t_i - w, t_i]$.

$$q_i = \arg\max_{1 \le j < i}(\hat{p}_1^{i,j} | \hat{p}_1^{i,j} > \hat{p}_2^{i,j} \wedge (t_i - t_j) \le w) \tag{6}$$

Then, as shown in Figure 6, if q_i exists, we add e_i into the incident of e_{q_i}. Otherwise, we form a new incident for e_i. Such strategy avoids modifying previous alerts and ensures that each alert is processed only once in the online summarizing stage.

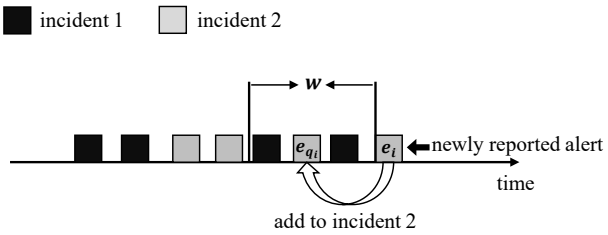


Figure 6: The process of alert summarizing.

8 EVALUATION

To evaluate the effectiveness of our approaches, we exploit real-world datasets from two large commercial banks to address the following research questions:

- RQ1: How does OAS perform in summarizing alerts?
- RQ2: How does the sample granularity α in ABR affect the summarizing performance?
- RQ3: How does the sample length β in ABR affect the summarizing performance?
- RQ4: How does the time window w in online summarizing affect the summarizing performance?

8.1 Datasets

We conduct experiments on real-world alerts from two large commercial banks, A and B. As shown in Table 2, alerts of Bank A and Bank B have different characteristics. The definition of the alert in Bank A is not strictly standardized, thus there are thousands of alert types in its dataset. In Bank B, the definition of the alert is quite rigorous, thus Bank B has a much smaller number of alert types than Bank A. Due to the information privacy, we can only share the alert dataset of Bank B, which is available at https://doi.org/10.5281/zenodo.5336985, in which all sensitive information is anonymized.

Table 2: Details of Experimental Datasets

Datasets	Time Span	#Alerts	#Alert Types
Bank A	2018/11/23 ~ 2019/02/02	50947	2794
Bank B	2019/03/01 ~ 2020/08/06	500000	51

8.2 Baselines

As shown in Table 3, we compare our approaches with seven approaches, SeqKrimp, GoKrimp, CSC, SWIFT, Jaccard, Word2Vec, and LDA. SeqKrimp, GoKrimp, CSC, and SWIFT are all frequent pattern mining approaches. They generically summarize an event sequence by mining frequent patterns that minimize the description length of the event sequence [24]. SeqKrimp [12], GoKrimp [12], and CSC [10] are offline approaches, while SWIFT [28] is an online approach. As frequent patterns reveal co-occurrence relations between alerts, we consider CSC, SeqKrimp, GoKrimp, SWIFT as a type of approaches that summarize alerts by behavior information.

Table 3: Information of Experimental Approaches

Approach	Semantic Information	Behavior Information	Online
SeqKrimp	×	√	×
GoKrimp	×	√	×
CSC	×	√	×
SWIFT	×	√	√
Jaccard	√	×	√
LDA	√	×	√
Word2Vec	√	×	√
ASR	√	×	√
ABR	×	√	√
OAS	√	√	√

Since Jaccard [13, 31], Word2Vec[19], and LDA [2, 33] are widely used to measure the semantic relevance of alerts, we thus compare our approaches with such three approaches. In addition, we also individually evaluate the ability of ASR and ABR, respectively. Specifically, to summarize alerts online by ASR, we adopt the online summarizing strategy in Section 7. For the newly generated alert, e_i, instead of ACT, we find its most relevant alert during $[t_i - w, t_i]$ by the cosine similarity between semantic representations. Then, if the maximum cosine similarity is larger than a fixed threshold, we then add e_i into the incident of the most relevant alert. Otherwise, we form a new incident for e_i. The same is true for ABR.

8.3 Setup

We implement ASR, ABR, ACT, OAS, Jaccard, LDA, and Word2Vec with Python 3.6 and PyTorch 1.5.1 [22]. For LDA and Word2Vec, we use a popular open-source NLP toolkit, Gensim [23]. As for SeqKrimp, GoKrimp, CSC, and SWIFT, we use a Java-based open-source toolkit [28]. All experiments are conducted on a 4.20GHz × 8 Intel i7-7700K PC with 16 GB memory running ubuntu. The source code of our approaches is available at https://doi.org/10.5281/zenodo.5998339.

In LDA, the number of topics is set to 9. For the CBOW model in ASR and Word2Vec, the size of the word embedding is 512 and the epoch is set to 100. For ABR, the size of the behavior representation is 600, the sample granularity for the occurrence series, α, is set to 1 minute, the sample length for the occurrence series, β, is set to 6 hours for Bank A and 13 hours for Bank B, and the epoch is set to 5. For ACT, in Linear Transformation, the first layer has 50 neurons, and the second layer has 30 neurons. In Dimension Reduction, the first layer has 20 neurons, and the second layer has 2 fixed neurons. The epoch of ACT is set to 60. The window size, w, in online summarizing, is set to 5 minutes. For approaches that require training, Table 4 demonstrates their time cost in the training stage. Although our approaches require the longest training time, they all take less than 10 minutes to finish training, which is actually acceptable since the training stage is conducted offline.

Table 4: Training Time of Experimental Approaches

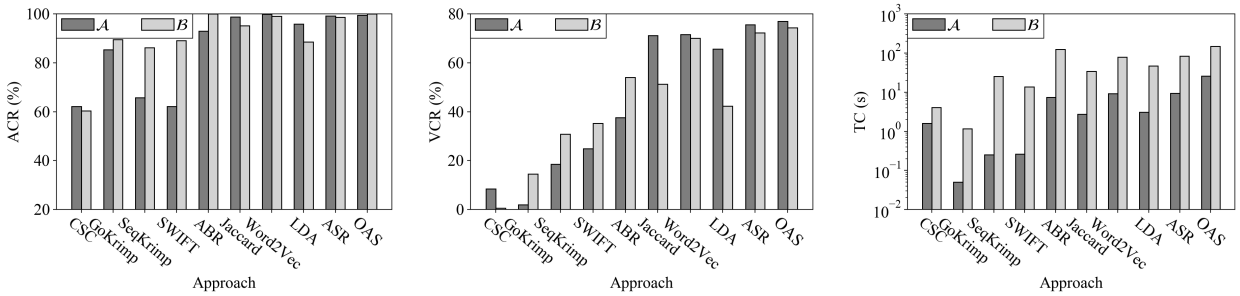
Bank	Training Time (s)				
	ASR	ABR	ACT	Word2Vec	LDA
Bank A	9.977	7.707	16.425	7.656	5.005
Bank B	24.667	330.388	260.671	24.658	57.214

Thresholds for experimental approaches are all adjusted to achieve the maximum valid compression ratio. For each dataset, according to the timestamp, we take the first 80% as the training data, and the last 20% as the testing data.

8.4 Metrics

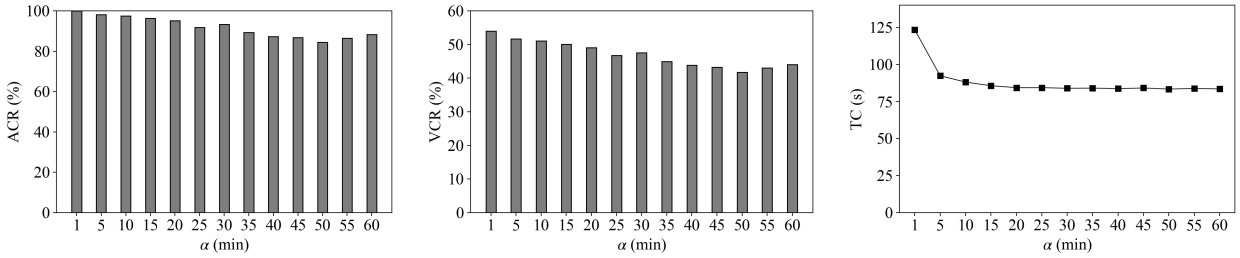
For a system failure, if an approach only groups a part of its triggered alerts into an incident, we do not consider the incident to be wrong, since the incident still groups correlated alerts correctly. Only when an incident contains an uncorrelated alert, we consider the incident to be wrong. After correctness verification, incidents can be divided into three groups, correct incidents, wrong incidents, and isolated incidents. An isolated incident contains only one alert and has no contribution to alert summarizing, thus there is no need to judge its correctness.

We use the incident accuracy (ACR), the valid compression ratio (VCR), and the time cost (TC) in summarizing stage to evaluate the performance of experimental approaches. The incident accuracy is defined as $ACR = \frac{N_c}{N_c+N_w} \times 100\%$, where N_c is the number of correct incidents and N_w is the number of wrong incidents. The incident accuracy represents the proportion of correct incidents in non-isolated incidents. The valid compression ratio is defined as $VCR = (1 - \frac{N_c+n_w+n_i}{n}) \times 100\%$, where n_w and n_i respectively indicate the number of alerts in wrong incidents and in isolated incidents. The valid compression ratio represents the true summarizing ability of the approach, and it ignores the contribution of wrong incidents and isolated incidents. The higher the summarizing ability of the approach, the higher the valid compression rate. Moreover, for each approach, we record its time cost (TC) in summarizing stage to evaluate its efficiency.



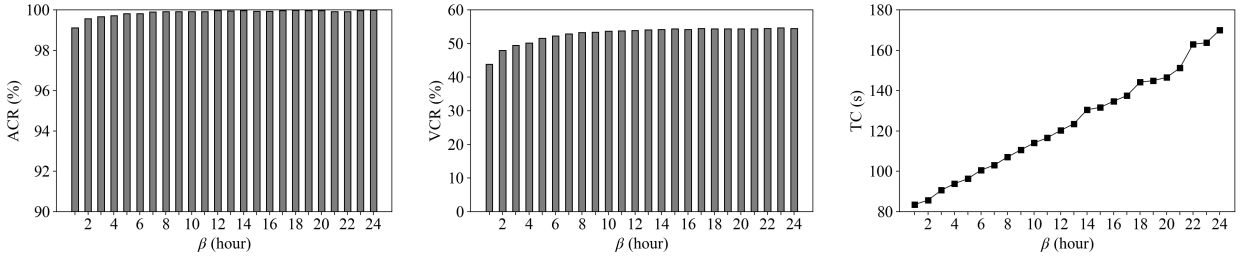
(a) Incident accuracy (*ACR*). (b) Valid compression ratio (*VCR*) (c) Summarizing time cost (*TC*)

Figure 7: Experimental results of summarizing performance.



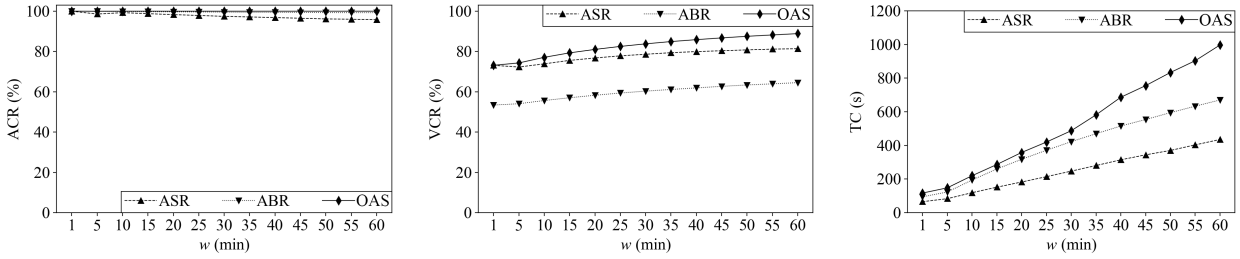
(a) Incident accuracy (*ACR*) (b) Valid compression ratio (*VCR*) (c) Summarizing time cost (*TC*)

Figure 8: Experimental results of varying sample granularity α in ABR.



(a) Incident accuracy (*ACR*) (b) Valid compression ratio (*VCR*) (c) Summarizing time cost (*TC*)

Figure 9: Experimental results of varying sample length β in ABR.



(a) Incident accuracy (*ACR*) (b) Valid compression ratio (*VCR*) (c) Summarizing time cost (*TC*)

Figure 10: Experimental results of varying time window w in online summarizing.

8.5 Evaluation Results

To answer the proposed research questions, we evaluate our approaches from four aspects, the summarizing performance of our approaches compared with the state of arts, the influence of the sample granularity α in ABR, the influence of the sample length β in ABR, and the influence of the time window w in online summarizing.

8.5.1 RQ1: summarizing performance. In the first experiment, we compare the performance of approaches for each dataset. Figure 7 shows the result. From Figure 7(a) and Figure 7(b), we can find that, in terms of the semantic information, ASR has higher *ACR* and *VCR* than Jaccard, Word2Vec, and LDA, in terms of the behavior information, ABR has higher *ACR* and *VCR* than CSC, GoKrimp, SeqKrimp, and SWIFT. Furthermore, OAS has the best performance among experimental approaches.

This is because ASR integrates the contextual information of words in an alert according to their semantic contributions. However, Jaccard only considers the number of the same words between alerts, ignoring the common semantics between different words, Word2Vec equally treats the words in an alert, neglecting varying contributions of words to the overall semantics, and LDA distills the general topic of an alert, failing to capture the distinctive semantics for alerts of the same topic. Moreover, ABR mines the commonality between alert occurrence series, while SeqKrimp, GoKrimp, CSC, and SWIFT try to find naive frequent patterns of alerts. Nevertheless, it is rare that the exact same failure recurs in a short period of time, thus correlated alerts may not have frequent co-occurrences.

Since OAS further has a better performance than ASR and ABR, the experiment demonstrates that both ASR and ABR have their own contributions to alert summarizing, and OAS can effectively aggregate the semantic information and behavior information by ACT. In addition, in both datasets, ASR contributes the most to alert summarizing. As show in Figure 7(c), to capture the deep semantic information and behavior information of the alert, ASR, ABR and OAS have higher *TC* in summarizing stage than other approaches. However, ASR, ABR, and OAS can still process the alert within acceptable time. In case of OAS, which has the maximum *TC* for each dataset, its *TC* for Bank A is 26s, which corresponds to more than 240 alerts per second, and its *TC* for Bank B is 147.73s, which corresponds to more than 670 alerts per second.

8.5.2 RQ2: varying sample granularity α in ABR. In this experiment, we evaluate the influence of the sample granularity, α, for the occurrence series in ABR. We vary α from 1 to 60 minutes. For each α, we train a new ABR model and evaluate its performance by the dataset of Bank B. Although there are some fluctuations in Figure 8(a) and Figure 8(b), which reveals that the relationship between α and summarizing performance is not strongly linear, a smaller α still tends to get better summarizing performance. Because when the sample length β is unchanged, a smaller α can generate a longer occurrence series, which is able to retain more detailed alert behavior information. As shown in Figure 8(c), since a larger α can generate a shorter alert occurrence series, ABR is more efficient with a larger α. However, even with the smallest α (1 minute), ABR still can process 100,000 alerts in about 2 minutes (more than 810 alerts per second). Thus, in conditions permit, we recommend choosing the smallest possible α.

8.5.3 RQ3: varying sample length β in ABR. As the sample length, β, in ABR determines the time range of the alert occurrence series, we evaluate its influence by varying β from 1 to 24 hours. Similar to the above, for each β, we train a new ABR model by the dataset of Bank B. From Figure 9(a) and Figure 9(b), we can find that when the sample granularity α is unchanged, as β increases, *ACR* and *VCR* also increase at first, and then they both gradually stabilize. A larger β results in a longer time range of the alert occurrence series, which contains richer behavior information. However, when β reaches a certain threshold, the time range of the alert occurrence series can already reveal the complete behavior information of the alert. More specifically, in Figure 9(a) and Figure 9(b), when $\beta > 13$ hours, *VCR* stabilizes at about 54%, while *ACR* is greater than 99%. Moreover, as shown in Figure 9(c), *TC* of ABR in summarizing stage has a linear relationship with β. When $\beta = 13$ hours, *TC* is about 2 minutes (more than 809 alerts per second). Therefore, we recommend choosing a small β that achieves a stable summarizing performance.

8.5.4 RQ4: varying time window w in online summarizing. Since the time window, w, in online summarizing controls the maximum time span between two correlated alerts, we thus evaluate its influence on ASR, ABR, and OAS by varying w from 1 to 60 minutes. Same as above, we adopt the dataset of Bank B as the experimental data. As shown in Figure 10(a) and Figure 10(b), OAS always has a better performance than ASR and ABR, proving that ACT can effectively integrate ASR and ABR. For ASR, as w increases, *VCR* decreases at first and then slowly increase until it stabilizes. Such phenomenon indicates a trade-off between the number of alert correlations in result incidents and the accuracy of result incidents. A small w can find a small number of alert correlations with high *ACR* while a large w can find a large number of alert correlations with low *ACR*. Nevertheless, for different w, both ABR and OAS have their *ACR* stable above 98%. Because ABR concentrates on the robust historical behavior information of alerts, and OAS integrates the strengths of ASR and ABR.

It should be noted that we use a five-minute w for the experiment of summarizing performance in Section 8.5.1, which seems not to be the best choice in Figure 10(b). This is due to that during the communication with front-line maintenance engineers, they propose that although a small w can lead to a high *ACR*, it also splits a failure into multiple fragmented incidents, and a large w is prone to merge independent failures of the same type into one incident. Both cases are detrimental to the failure analysis work of maintenance engineers. According to the experience of maintenance engineers, they present that 5 minutes is usually the maximum interval between two correlated alerts for their bank service systems. In addition, for ASR, ABR, and OAS, there is a linear relationship between w and *TC*. Thus, to choose the the time window w, we recommend that both experimental results and the practical maintenance experience should be taken into consideration.

8.6 Threats to Validity

We identify following threats to validity in our study.

Noises in labeling: We obtain alert correlations from history failure reports written by domain experts. A failure report records the root cause and the impact of a failure, as well as correlated alerts. Since correlated alerts in a failure report are manually labeled by experts, mislabeling is hard to avoid during the manual process. However, as our experts are all front-line maintenance engineers with rich domain knowledge, we are confident that the amount of noises in labeling is small (if it exists).

Generalizability: In our experiments, we use alerts from systems of two large commercial banks, A and B. The performance of our approaches depends on the number of alerts and the quality of alert correlation data (failure reports). In our study, Bank A and Bank B have accumulated many historical alerts, and their engineers are required to write failure reports, which we use to label the alert correlation. Similarly, in most commercial companies, both historical alerts and failure reports are maintained for a long time. In addition, Bank A and Bank B are two very large-scale banks, providing service for more than a billion users from hundreds of countries. Thus, we believe our experimental results can demonstrate the value of our proposed approaches and our approaches can be generalized to alerts of other companies.

Measurements: To demonstrate the summarizing performance of our approaches, we use *ACR* (Accuracy) and *VCR* (Valid Compression Ratio) as measurements, which are defined in Section 8.4. Although F-measure is a widely used measurement, it is unsuitable for our study. Because the correctness verification in F-measure is dichotomous. However, in alert summarizing, if an incident (not isolated) contains no uncorrelated alerts, then even if it does not contain all alerts of its corresponding failure, it is still correct in practical production. Therefore, after a discussion with domain experts, we propose *ACR* and *VCR* to measure the effectiveness of our approaches. *ACR* measures the proportion of correct incidents, and it ignores the impact of isolated incidents that only contain one alert. *VCR* measures the ability of mined incidents to summarize alerts, and it ignores the contribution of wrong incidents and isolated incidents to summarizing.

9 LESSONS LEARNED

In our study, after we cooperated with some commercial companies, we found that the alert management system of a large online service system needs a top layer or global design. First, the top layer design is useful to combine alerts from various tool, like Zabbix [16], Prometheus [6] and etc. Second, top layer design is helpful to avoid the monitoring blind spot. We can not find the root cause if it is not monitored. Moreover, we found that many large commercial companies have accumulated a large number of failure reports (or work orders). In our study, we use the knowledge in these failure reports to summarize alerts. However, we suggest that these failure reports can not only be used to correlate alerts, but also be considered as a knowledge base. These failure reports contain valuable expert experience, such as the root cause of the failure, the alert correlation, and the troubleshooting process. Therefore, these failure reports should have a more important role to be explored.

10 CONCLUSION

In this paper, we propose a framework OAS to efficiently summarize alerts online. In OAS, to represent the alert semantic information, we present ASR, which aggregates the contextual information of alert words according to their semantic contributions. To represent the alert behavior information, we present ASR, which captures the commonality between the occurrence series of alerts. In addition, we present a novel model, ACT, to combine the semantic and behavior information of the alert, and summarize the newly reported alert online by a time window. We conduct extensive experiments on real datasets from two large commercial banks, and the result demonstrates that our approaches outperform the state of the art. In future, we plan to provide the ability for OAS to suggest the root cause of the system failure based on the mined alert information.

REFERENCES

[1] Amey Agrawal, Rohit Karlupia, and Rajat Gupta. 2019. Logan: A Distributed Online Log Parser. In *IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1946–1951.

[2] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.

[3] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An Empirical Investigation of Incident Triage for Online Service Systems. In *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE, 111–120.

[4] Yujun Chen, Xian Yang, Qingwei Lin, Hongyu Zhang, Feng Gao, Zhangwei Xu, Yingnong Dang, Dongmei Zhang, Hang Dong, Yong Xu, Hao Li, and Yu Kang. 2019. Outage Prediction and Diagnosis for Cloud Service Systems. In *The World Wide Web Conference*. ACM, New York, NY, USA, 2659–2665.

[5] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1285–1298.

[6] The Linux Foundation. 2022. *Prometheus*. Retrieved February 5, 2022 from https://prometheus.io

[7] Shangbin Han, Qianhong Wu, Han Zhang, Bo Qin, Jiankun Hu, Xingang Shi, Linfeng Liu, and Xia Yin. 2021. Log-Based Anomaly Detection With Robust Feature Extraction and Online Learning. *IEEE Transactions on Information Forensics and Security* 16 (2021), 2300–2311.

[8] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 33–40.

[9] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2018. Identifying Impactful Service System Problems via Log Analysis. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, NY, USA, 60–70.

[10] A Ibrahim, Shivakumar Sastry, and PS Sastry. 2016. Discovering compressing serial episodes from event sequences. *Knowledge and Information Systems* 47, 2 (2016), 405–432.

[11] Guofei Jiang, Haifeng Chen, Kenji Yoshihira, and Akhilesh Saxena. 2009. Ranking the Importance of Alerts for Problem Determination in Large Computer Systems. In *Proceedings of the 6th International Conference on Autonomic Computing*. ACM, New York, NY, USA, 3–12.

[12] Hoang Thanh Lam, Fabian Mörchen, Dmitriy Fradkin, and Toon Calders. 2014. Mining compressing sequential patterns. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 7, 1 (2014), 34–52.

[13] Derek Lin, Rashmi Raghu, Vivek Ramamurthy, Jin Yu, Regunathan Radhakrishnan, and Joseph Fernandez. 2014. Unveiling Clusters of Events for Alert and Incident Management in Large-Scale Enterprise It. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 1630–1639.

[14] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log Clustering Based Problem Identification for Online Service Systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, New York, NY, USA, 102–111.

[15] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. 2020. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. In *IEEE 31st International Symposium on Software Reliability*

Engineering (ISSRE). IEEE, 48–58.
[16] Zabbix LLC. 2022. *Zabbix*. Retrieved February 5, 2022 from https://www.zabbix.com
[17] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.
[18] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, and Rong Zhou. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI Organization, 4739–4745.
[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at International Conference on Learning Representations (ICLR)*. 1–12.
[20] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. 2019. Anomaly Detection and Classification using Distributed Tracing and Deep Learning. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 241–250.
[21] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. 2019. Anomaly Detection from System Tracing Data Using Multimodal Deep Learning. In *IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 179–186.
[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035.
[23] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50.
[24] Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica* 14, 5 (1978), 465–471.
[25] Roel Wieringa. 2010. Design Science Methodology: Principles and Practice. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering* (Cape Town, South Africa). ACM, New York, NY, USA, 493–494.
[26] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. 2018. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In *Proceedings of the 2019 World Wide Web Conference*. ACM, Republic and Canton of Geneva, CHE, 187–196.
[27] Jingmin Xu, Yuan Wang, Pengfei Chen, and Ping Wang. 2017. Lightweight and Adaptive Service API Performance Monitoring in Highly Dynamic Cloud Environment. In *IEEE International Conference on Services Computing (SCC)*. IEEE, Los Alamitos, CA, USA, 35–43.
[28] Yizhou Yan, Lei Cao, Samuel Madden, and Elke A. Rundensteiner. 2018. SWIFT: Mining Representative Patterns from Large Event Streams. *Proc. VLDB Endow.* 12, 3 (Nov. 2018), 265–277.
[29] Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping Qu, and Lei Song. 2017. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.
[30] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. Robust Log-Based Anomaly Detection on Unstable Log Data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, NY, USA, 807–817.
[31] 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 *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. ACM, New York, NY, USA, 162–171.
[32] Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Real-Time Incident Prediction 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*. ACM, New York, NY, USA, 315–326.
[33] Nengwen Zhao, Panshi Jin, Lixin Wang, Xiaoqin Yang, Rong Liu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Automatically and Adaptively Identifying Severe Alerts for Online Service Systems. In *IEEE Conference on Computer Communications*. IEEE, 2420–2429.
[34] 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 Conference on Computer Communications*. IEEE, 1882–1890.
[35] Nengwen Zhao, Jing Zhu, Yao Wang, Minghua Ma, Wenchi Zhang, Dapeng Liu, Ming Zhang, and Dan Pei. 2019. Automatic and Generic Periodicity Adaptation for KPI Anomaly Detection. *IEEE Transactions on Network and Service Management* 16, 3 (2019), 1170–1183.
[36] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, NY, USA, 683–694.