IEICE TRANSACTIONS

on Communications

DOI:10.1587/transcom.2018EBP3103

Publicized:2018/08/13

A PUBLICATION OF THE COMMUNICATIONS SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Proactive Failure Detection Learning Generation Patterns of Large-scale Network Logs

Tatsuaki KIMURA[†], **Akio WATANABE**[†], **Tsuyoshi TOYONO**[†], ***Members***,
and **Keisuke ISHIBASHI**[†], ***Senior Member***

SUMMARY Recent carrier-grade networks use many network elements (switches, routers) and servers for various network-based services (e.g., video on demand, online gaming) that demand higher quality and better reliability. Network log data generated from these elements, such as router syslogs, are rich sources for quickly detecting the signs of critical failures to maintain service quality. However, log data contain a large number of text messages written in an unstructured format and contain various types of network events (e.g., operator's login, link down); thus, genuinely important log messages for network operation are difficult to find automatically.

We propose a proactive failure-detection system for large-scale networks. It automatically finds abnormal patterns of log messages from a massive amount of data without requiring previous knowledge of data formats used and can detect critical failures before they occur. To handle unstructured log messages, the system has an *online log-template-extraction* part for automatically extracting the format of a log message. After template extraction, the system associates critical failures with the log data that appeared before them on the basis of supervised machine learning. By associating each log message with a log template, we can characterize the generation patterns of log messages, such as burstiness, not just the keywords in log messages (e.g. `ERROR, FAIL`). We used real log data collected from a large production network to validate our system and evaluated the system in detecting signs of actual failures of network equipment through a case study.

key words: *Syslog, network management, failure detection, machine learning*

1. Introduction

Various network-based services (e.g. video on demand, online gaming) deployed in current carrier-grade networks demand much higher network quality and reliability than previous services. Network operators need to replace their current *reactive* operations with *proactive* operations for service providers in fierce competition. They need to detect even a minor temporal event, such as a minute internet-disconnection, because it may lead to service providers suffering significant losses. In general, network operators start troubleshooting when they detect failure alarms from network management systems (NMSs) on the basis of predefined alarm rules or customer complaints. However, predefined alarm rules often represent obvious critical statuses and fail to detect minor temporary events. As a result, operators can *react* to network problems only after a service has been seriously affected. Unfortunately, such reactive operation is no longer acceptable because of the very high requirements of network services.

Syslogs of network elements and alert logs generated in NMSs are important information for current network operation. A syslog is a standard for message logging [1], and network elements commonly store syslog messages or send them to NMSs for troubleshooting. Some alert messages are generated on the basis of predefined rules for the syslog messages and include Simple Network Management Protocol (SNMP) trap messages (e.g., 'ping timeout'). Table 1 lists examples of logs messages. These logs contain detailed information of network elements: not only critical hardware failures but also reports on the normal statuses of various protocols and layers. Network operators must monitor log messages on a daily basis.

However, genuinely important log messages related to major network problems have become difficult to find. First, the log messages generated in a carrier-grade network consist of many text messages written in unstructured formats. Since modern networks have tens of thousands of network elements, the total volume of log messages is tens of millions per day. In addition, network elements consist of multiple vendors and have various rules in multiple services. Since the format of logs varies with the vendor or service, there are also numerous types of log messages. As a result, it has become time consuming for network operators to check all the massive log data and find small preliminary signs of critical failures.

Second, log messages are highly diverse because they contain various types of network events ranging from critical hardware failures to normal login events of operators. It thus requires extensive experience and knowledge to determine the meanings of log messages. Network operators generally carry out rule-based monitoring of log messages to detect failures by using a severity level described in a message or setting keywords filters to messages, e.g. `ERROR, FAIL`. However, such rule-based monitoring is often insufficient. This is because the severity measure may not be obtained for some vendors, and the predefined keywords are not always included in messages indicating the temporally abnormal status that leads to future failures. Furthermore, the occurrences of the same log messages do not always represent the same statuses of network elements. For example, a sudden burst of a log message that usually occurs alone can be considered abnormal.

We propose a log-analysis system for proactive failure detection. The system automatically learns the relationship

[†]The authors are with the NTT Network Technology Laboratories, NTT Corporation, Mushishino-shi Tokyo, 180-8585 Japan.

Table 1 Examples of log messages

#	timestamp	host	messages
1	2015/1/1T00:00:00	HOST_X	%TRACKING-5-STATE: 1 interface Fa0/0 line-protocol Up− >Down
2	2015/1/1T00:00:00	HOST_X	%LINK-3-UPDOWN: Interface FastEthernet 0/0, changed state to down
3	2015/1/1T00:00:05	10.1.1.2	%SYS-5-CONFIG_I: Configured from console by vty0 (10.1.1.2)
4	2015/1/1T00:00:10	HOST_Y	100 login : LOGIN_INFORMATION : User XXX logged in from host HOST_X on device X
5	2015/1/1T00:00:11	HOST_Y	chassisd [111] : CHASSISD_BLOWERS_SPEED : Fans and impellers are now running at normal speed
6	2015/1/1T00:00:15	10.1.1.3	SNMP trap: CPU utilization exceeds threshold (96.9 % > 90 %)
7	2015/1/1T00:01:00	10.1.1.3	Ping Timed Out (6 / 6)

between critical failures and log messages that appeared before them without requiring any previous knowledge of log data formats. By noticing such anomalies in advance, network operators can carry out proactive operations such as preventive maintenance or arrangement of spares. In addition, time-consuming log checking is no longer needed. To use the rich features of the vast amount of complex log data, we developed an online template-extraction method for our system. It can automatically and quickly transform log messages into *log templates* by performing online clustering on the basis of the similarity of log messages. The log templates are messages without parameters, such as IP addresses, and hostnames, (i.e. the primary components of log messages). By extracting the log templates, we can characterize the generation patterns of log messages, such as frequency, periodicity, and burstiness. To automatically associate the critical failures with the log data, we take a supervised machine learning approach by using the generation patterns of log messages as a feature vector and trouble ticket data as training datasets. Our key observation is that the abnormality of logs depends on not the keywords in log messages but these log-generation patterns. For example, frequent messages, such as firewall logs, and periodic messages, such as `cron` jobs, can be considered as normal. On the other hand, there are log messages that can lead to failures when they occur in sudden bursts even though they do not affect the network when they occur alone. We validated our system by using a massive number of log data collected from a large-scale production network. The experimental results indicate that our system can detect abnormal network statuses before critical failures occur and achieve much better classification than current monitoring systems.

We briefly summarize our contributions as follows:

- We developed a machine-learning-based proactive failure-detection system on the basis of the generation patterns of log data associated with log templates.
- We developed an online template-extraction method for our system that can automatically and quickly extract log templates from massive and unstructured log data with high accuracy.
- We validated our system using a massive amount of log data collected from a large production network through comprehensive experiments. The results indicated that the proposed system has better detection performance than the current keyword-based monitoring system.

The rest of this paper is organized as follows. Section 2 summarizes related work. In Section 3, we explain our proposed system in detail. In Section 4, we discuss several experiments that we conducted for evaluating our system. Finally, we conclude the paper in Section 5.

A part of this research was presented at a conference [17].

2. Related Work

Commercial products There are many commercial products for complicated network management and operations, e.g., [2]–[6]. NMSs import various metrics of network elements (such as traffic) and CPU utilization of routers, visualize them, and raise alarms on the basis of predefined rules, e.g., keywords and severity. However, as mentioned earlier, these predefined rules often indicate apparently critical statuses and cannot detect the temporal abnormal statuses that may cause serious problems in the future. Splunk [6] is a platform intended to improve the use of syslogs and other types of logs. It collects log data, makes indexes, visualizes logs, and helps in the fast analysis of data. However, efficiently using these commercial products requires extensive experience and domain knowledge such as rule finding and log-format definition.

Log analysis Machine-generated log data have been extensively investigated. Yamanishi et al. [39] proposed a technique to detect system failures from server syslogs using a mixture of hidden Markov models. Lim et al. [22] proposed a system-log mining method using only the frequency of log messages. Zheng et al. [42] introduced a log-preprocessing method of filtering important logs. Xu et al. [37] analyzed console logs of large-scale hadoop systems and proposed a principal component analysis (PCA)-based anomaly-detection method. Vaaradi and Podiņš [35] proposed a classification method of network IDS alerts via a frequent item set mining approach. Fu et al. [13] introduced an anomaly-detection technique by learning normal system behaviors with a finite state automaton. The above studies basically focused on anomaly detection in an unsupervised manner, i.e., aimed at classifying whether the current status is abnormal by detecting a change in the relationship between performance metrics learned from historical data. On the other hand, we focused on finding signs, i.e., *proactive detection* of future failures, in a supervised approach with

trouble-ticket data. In other words, we aim to reveal the hidden relationship between critical failures and log data occurring before the failures. We also show that generation patterns of logs can be used for such a proactive failure-detection problem.

SyslogDigest [26] targets the router syslogs in a Tier-1 network. However, it just constructs digest information by grouping log messages within relevant routers. Kimura et al. [18] proposed a modeling and event-extraction method of network log data using a tensor-factorization approach. Furthermore, the correlations and causal relationship between network events were analyzed by González et al. [14] and Kobayashi et al. [20]. However, they did not focus on proactive failure detection. The most closely related work is that by Sipos et al. [29]. They presented a multiple-instance learning approach for predicting equipment failures by analyzing system-log data. Most recently, Zhang et al. [41] proposed a deep-learning-based failure-detection system. Unlike us, they did not use the generation patterns of log data such as burstiness. In addition, Sipos et al. [29] did not use automatic template extraction and used *message types* described in their data. Similarly, Reidemeister et al. [28] studied recurrent failure detection from unstructured logs on the basis of a decision-tree classifier but did not use these features.

Data-driven network management Many studies have also been conducted in the area of network management using various network data. For example, NICE [23] extracts the statistical correlation between temporal network events with time lags using various data collected at a Tier-1 network (e.g., traffic, CPU utilization). On the other hand, Orion [10] extracts the causal relationship among application traffic by analyzing each traffic delay in an enterprise network. In addition, eXpose [15] reveals flow-generation rules by learning the association rules of flows from packet trace.

Root-cause analysis and fault localization are also important topics in this research area. Sherlock [9] learns a dependency graph between multilevel resources in enterprise networks for analyzing the root causes of failures. NetMedic [16] monitors behaviors of applications and conducts detailed diagnoses of fault events occurring in enterprises. TAR [32] localizes faults and automatically learns fault events from the event datasets by indexing the network events. G-RCA [40] is a root-cause analysis system that identifies the root cause of a problem by matching a current event to a predefined decision tree, which represents the causal relationships of the network events. However, the above systems use the structured network event data collected by NMSs, whereas our system extracts such events from raw primitive log data and detects unknown failure events.

Another line of this research area is an approach with trouble-ticket data. NetSieve [25] extracts useful information and infers root causes using trouble tickets. Relatively recently, Watanabe et al. [33] proposed a work-flow mining technique from trouble-ticket data, which automatically constructs a flow of operator's actions during a troubleshooting process. In our research, we used trouble-ticket data as training data of supervised machine learning, so trouble-tickets analysis was out of the scope of our research.

3. Proposed System

In this section, we explain our proposed proactive failure-detection system based on the generation patterns of logs data.

3.1 System Overview

The main objective with our system is to automatically learn the relationship between critical future failures and log data and proactively detect an abnormal pattern of log messages that will lead to future problems by analyzing recent logs. Fig 1 gives an overview of our system. The system first splits log messages by using a certain time window (e.g. 15 or 30 min.) and groups them into chunks with the same hosts, called **log chunks**. For a given log chunk, our system classifies whether the current status may lead to critical problems and reports it to network operators. Since the input of our system, a log chunk, is a snapshot of observed log messages, network operators just put the arriving log messages into the system as they are and do not need any knowledge of log messages. To achieve this goal, the system first preprocesses unstructured logs and transforms each message into a log template that enables the system to characterize the log-generation patterns (Section 3.2). The system extracts static metrics, such as frequency of log templates, offline for constructing the log features later. The newly extracted log templates and these metrics are stored in an offline database. The system then creates log feature vectors that characterize the generation patterns of log messages for each log template in a log chunk (Section 3.3), and the extracted feature vectors are aggregated within a chunk (Section 3.4). Finally, the machine learning classifier detects future failures for the aggregated log feature vector (Section 3.5). The classifier model is trained offline using network trouble-ticket data. We now explain each part of our system in detail.

3.2 Online Log-Template Extraction

Network logs include various types of messages ranging from critical failure to normal console logs. As shown in Table 1, log messages typically consist of three components: (i) a timestamp, (ii) host identifier (e.g. IP addresses), and (iii) message. We can see from the table that the format of log messages depends on vendors or services, and there is no unified rule for description. This is why it is time consuming to describe regular expressions and define new alert rules for each message. Furthermore, it is unrealistic to use raw messages directly to extract features that can be used for proactive failure detection, such as generation patterns of log messages. More specifically, messages with error IDs or process IDs may never appear twice depending on the uniqueness of these IDs. Therefore, we need to focus not on log messages but log templates. We give example log

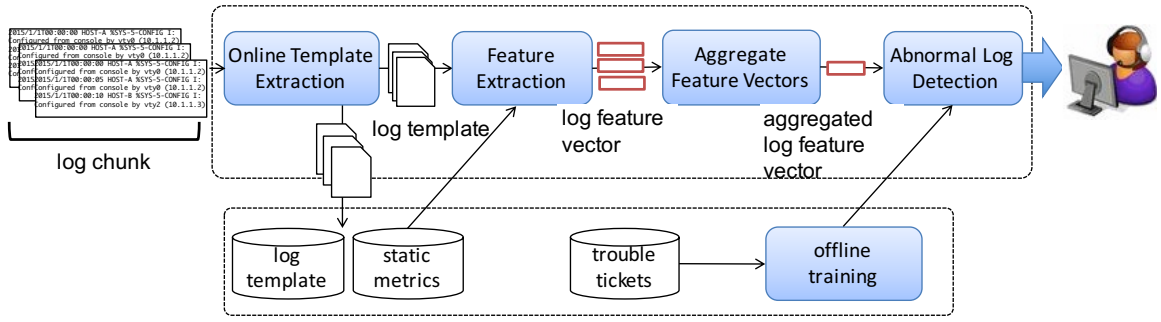


Fig. 1 System overview

templates in Fig. 2. These log templates can be obtained from vendors' support pages or manuals. However, the formats may change due to OS upgrades or maintenance, and not all descriptions can be obtained.

Due to the above reasons, template-extraction methods have been proposed e.g., [18], [24], [26], [44]. However, many such methods, such as those proposed by Kimura et al. [18] and Qiu et al. [26], are offline batch schemes; thus, require a large amount of training data to learn log templates. In addition, the format of messages may dynamically change in the future because of software upgrades. Contrary to offline template extraction, a few researchers proposed incremental template-extraction methods [24], [44]. However, the method proposed by Zhu et al. [44] basically takes a mini-batch learning approach based on an offline template-extraction method, so it is not a complete online method. On the other hand, Mizutani [24] proposed an incremental template-extraction method based on online clustering of log messages. However, the clustering method uses a simple similarity score between log messages, which calculates the similarity between words and may not detect the slight similarity between log messages, such as a single character '**:**' or '**;**'.

For this problem, we developed an online template-extraction method, which can learn log templates incrementally with little computational cost. The main ideas for the method are: (i) classification of each word in a log message on the basis of the tendency to belong to a log template; (ii) online clustering of arriving messages by regarding a log template as a cluster of messages and calculating the *log similarity* between a cluster and log message on the basis of the classes of words; and (iii) the optimization of weight parameters in a log similarity score. We explain the key features of our method below.

(i) **Classification of words** From the observation of log messages, symbol words, such as "=" or ":", are likely to belong to log templates; on the other hand, numerical words, such as process IDs, can be considered parameters. In accordance with this idea, we first classify the words in the sense of the tendency to belong to a log template. The detailed definition of classification of words is described in Table 2. Furthermore, we define $\boldsymbol{w} = [w_i]$ $(i = 1, 2, \ldots, 5)$ as a weight vector that corresponds to the tendency to become log templates for each class i.

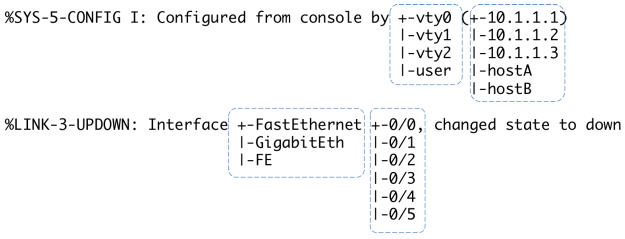


Fig. 2 Template clusters. Words in box represent parameter words, and newly arriving one will be stacked in same positions.

Table 2 Classes of words

class	definition	examples
1	only numbers or numbers and symbols	1, 0/0, 10.1.1.1
2	numbers and letters	host-01, IPv4, L2TP, vty0, Fa0/0
3	symbols and letters	class-a, udp-port, aaa.cfg, line-protocol
4	only letters	linkdown, state, interface
5	only symbols	<, >, =, :

(ii) **Online message clustering** Next, for each arriving log message, we carry out online clustering so that the message is assigned to the cluster (called a template cluster) with the highest similarity in a similar manner to that of Mizutani [24]. To do this, we define the following log similarity between a cluster C and message X as

$$LogSimilarity(C, X) = \boldsymbol{w}^{\mathrm{t}}\boldsymbol{x}/\boldsymbol{w}^{\mathrm{t}}\boldsymbol{c}_x,$$

where $\boldsymbol{x} = \{x_i\}$ represents the number of class i words in X and $\boldsymbol{c}_x = \{c_{x,i}\}$ represents the number of class i words that appeared in both C and X. In short, the above log similarity represents a co-occurrence measure of words weighted by the classes of words. If the highest log similarity is less than a predefined threshold E, then we create a new template cluster from X.

```
1: template cluster set C = ∅;
2: for each message X do
3:    GetWordClass(X)
      C := FindNearnestCluster(C, X);
4:    if LogSimilarity(C, X) ≥ E then
5:       append X to cluster C;
6:    else
7:       create a new cluster from X;
         C := C ∪ {X};
8:    end if
9: end for
```

Fig. 3 Online template-extraction pseudocode

(iii) **Parameter optimization** Since the accuracy of template clustering depends on the weight parameter $\boldsymbol{w}$ and E, we need to optimize them. From the definition of log similarity, we can consider the problem of assigning a log message to a cluster as a linear classification problem such that

$$\mathrm{sign}(\boldsymbol{w}^{\mathrm{t}}[\boldsymbol{x} - E\boldsymbol{c}_x]) \begin{cases} \geq 0, & \text{assign } X \text{ to } C, \\ < 0, & \text{create a new cluster.} \end{cases}$$

Thus, by feeding back the result of whether the message is correctly assigned to the template cluster or separated, we can update and optimize $\boldsymbol{w}$. In our experiment discussed in Section 4, we chose PA-I [11] as the learning algorithm, which is a well-accepted online supervised classifier.

Fig. 3 shows the pseudo-code of online template extraction. Our method first extracts the classes of words of an arriving X. It then searches for the nearest template cluster among those with the same length and symbols. If the log similarity is larger than E, X is aggregated to the cluster; otherwise, a new cluster is created from X. From the definition of E, if E takes a larger value, our method tends to split clusters more aggressively; otherwise, template clusters tend to be aggregated with each other.

3.3 Feature Extraction

After template extraction, our proposed system attempts to obtain the features of log templates in each log chunk to characterize the generation patterns. At the same time, the static metrics of these features, such as frequency of log templates, are stored in an offline database. Suppose that log templates in a certain log chunk at a host h are $\{t_1, \ldots, t_N\}^{\dagger}$. We then create a log feature vector $\boldsymbol{x}_i$ $(i = 1, \ldots, N)$ for each log template $\{t_i; i = 1, \ldots, N\}$.

The simplest approach to construct a log feature vector is a *bag-of-words* expression of words used in natural language processing and information retrieval areas. In this expression, each element of a feature vector represents the existence or the number of words in a log template. However, this approach has a similar problem with keyword-based monitoring: words seemingly about abnormalities, e.g. `ERROR`, `DOWN`, are not always related to problems. This is partially because syslog messages are basically designed as debug messages by vendors. For example, a user session-disconnection event causes messages with words `DISCONNECTION` or `DOWN`. However, this occurs throughout the network on a daily basis. Therefore, we adopted as features not words in log messages but the generation patterns of log messages such as frequency, periodicity, or burstiness. For instance, periodic log messages, such as log messages induced by cron jobs and Internet Control Message Protocol (ICMP) and SNMP polling can be considered normal. Furthermore, if the log message appears only once, then it can be ignored, although if it appears in a burst, a replacement is needed. In accordance with these observations, we select the following features from log-generation patterns.

(a) **Frequency** The first basic feature is the frequency of log templates. Typically, frequent log messages, such as firewall logs and user connection/disconnection messages, can be considered normal. On the other hand, operators need to check infrequent messages. We simply count the number of occurrences of log templates during the observation period and use it as the frequency feature after normalization.

(b) **Periodicity** There are periodic log messages, such as messages induced by cron jobs, ICMP and SNMP polling, and daily maintenance operations. Although these periodic log messages may be infrequent, they are unlikely to be related to failures. In contrast, if there are certain biases with the occurrence of a log message, the message is not generated by a routine operation. Since a wide range of granularity for the period can be considered, we define the periodicity of log templates as the coefficient of variation in the observed number of log templates within each *hour, day, week, and month*. Formally, for each log template t, let

$$Periodicity(t, ITVL) = \frac{\sigma_{t,ITVL}}{\overline{D}_{t,ITVL}},$$

where $ITVL$ represents an interval (i.e., an hour, day, week, or month), and $\overline{D}_{t,ITVL}$ and $\sigma_{t,ITVL}$ are the mean and standard deviation of the observed number of log templates within the intervals. Although there are many candidates for calculating periodicity, such as Fourier transform, the reason we choose this simple metric is that even if the events the messages signify are the same, the intervals of log messages are varied. This is because manual daily operations are not strictly periodic (e.g. some are done in the morning and others in the evening).

(c) **Burstiness** Some log messages become failures when they occur in sudden bursts, although the message itself is not critical when it appears alone. For example, a single bit error at a certain module will be fixed by its error-correction circuit and will not affect the network.

[†]Note that N does not equal the total number of the log templates observed during the whole period. We consider a fixed log chunk and N represents the number of the log templates in it.

However, if the bit error occurs more frequently than before, the module may potentially crash and should be replaced (see e.g. Cisco's support page [8]). Therefore, burstiness of log messages is an important feature for discovering signs of future failures.
To calculate bursty features, we simply adopt Kleinberg's burst-detection algorithm [19]. Briefly, this algorithm models the occurrence of an event using a finite state hidden Markov model, in which each hidden state corresponds to a different Poisson process with a different parameter. It learns automatically from which hidden state each event is emitted. The state with a higher Poisson parameter indicates the burst state. Thus, we use this 'burst level' as a bursty feature.
Since Kleinberg's algorithm calculates the burstiness in an unsupervised manner, a baseline for the interval times of log messages needs to be contained in the input data, i.e., the interval time of log messages generated when they are in normal status. Thus, we combine an arriving log chunk with *dummy log messages*, which are artificially generated log templates that use the mean value of the interval time of each log template. Since the dummy log messages correspond to those generated when they are in normal status, the algorithm can detect the baseline of interval time and determine the correct burstiness for the current states (current log chunk). These mean values for all log templates are stored in an offline database as static metrics extracted during the log-template extraction.

(d) **Correlation with maintenance and failures** In an actual production network, numerous maintenance operations occur throughout the network daily. These operations cause various log messages and sometimes confuse network operators because it is difficult to determine which log message was caused by maintenance. To take into account the tendency of log messages to appear during maintenance, we add the correlation to the maintenance feature calculated as

$$\frac{\text{(\#of log templates observed during maintenances)}}{\text{(\#of log templates observed during the whole period)}}.$$

To calculate the above value, we use maintenance procedure data, which describe what kind of maintenance is performed when and on which host. Similarly to maintenance, we calculate the correlation to failures by using trouble-ticket data. This feature represents the tendency of log templates to appear during failures.

We calculate the above features for each log template $\{t_i; i = 1, \ldots, N\}$ in the target log chunk. Features (a) and (b) are also counted for each tuple (host, log template), i.e., $\{(h, t_i); i = 1, \ldots, N\}$, to take into account the host-specific information. By combining them, we finally create a log feature vector $\boldsymbol{x}_{t_i}$ for each log template.

3.4 Feature Aggregation

Next, our system aggregates log feature vectors and obtains a single vector that fully characterizes a log chunk. After extracting log feature vectors from a log chunk, we obtain the set of log feature vectors $\{\boldsymbol{x}_{t_1}, \ldots, \boldsymbol{x}_{t_i}, \ldots, \boldsymbol{x}_{t_N}\}$ corresponding to log templates t_i $(i = 1, \ldots, N)$ in a log chunk. To apply a binary classification problem, the log feature vectors at host h are aggregated into an aggregated log feature vector $\boldsymbol{x}_h$ that has the same dimensions as $\{\boldsymbol{x}_{t_i}\}$. By merging all information in a log chunk, the aggregated feature vector can efficiently represent the status of the current log chunk such as the existence of bursty messages or many periodic logs. Since each element of a log feature vector has a different character, we apply a different aggregation scheme for each element of a feature vector.
Mean: The simplest approach to form an aggregate vector is to calculate a mean value among all vectors. We adopt 'mean' operator for frequency, periodicity, correlation to failures, and keywords features.
Max: For the bursty feature, the important information is whether a bursty log template is in a log chunk. Therefore, we calculate the maximum value of the burst levels.
Min: If the correlation value for maintenance is large, the system should ignore the log template. Thus, we adopt a 'minimum' operator to extract the element mainly related to failures.

3.5 Machine-Learning-Based Proactive Failure Detection

As a final part of our system, the system determines whether the current status leads to a future problem for each aggregated feature vector $\boldsymbol{x}_h$. To detect future failures accurately, we adopt a supervised machine-learning approach using network trouble-ticket data. More precisely, we consider a binary classification problem in which given the h-th failure label $y_h \in \{1, -1\}$ (1 is the failure state in the future, called *positive*, and -1 is the normal state, called *negative*) and corresponding feature vectors $\boldsymbol{x}_h$, we classify an unlabeled feature vector into binaries. In our research, we adopted a support vector machine (SVM) with the Gaussian kernel [12] for supervised machine learning. An SVM is a well-accepted tool for binary classification. In short, it constructs a hyperplane that maximizes the margin between two classes corresponding to the labels.

4. Experiments

In this section, we discuss our experimental results for using real log data for both online template extraction and future failure detection.

4.1 Online-template-extraction evaluation

We first explain the evaluation results for the online-template-extraction part presented in Section 3.2. To do this, we set two metrics: (i) accuracy of template extraction; and (ii) performance of the template extraction algorithm. In the following experiments, we selected the best weight parameter $\boldsymbol{w}$ on the basis of a prior experiment.

Table 3 Examples of BlueGene log messages [7]

MSG_ID	messages
KERN_080A	controller 1, chipselect 0 single symbol error count 1
KERN_0115	instruction cache parity error has occurred in TAG bit 1 line = 0x00000217 word = 0x00000001
KERN_0804	chipkill error: DDR Controller 1, failing SDRAM address 0x01fffe9a0, chipkill location 0x100, either X8 compute DRAM chip U31 or U11.
KERN_1014	sender retransmissions: Node (31 , 4 , 4) had 1 retransmissions in the X+ direction

Accuracy of template extraction: Due to the lack of ground-truth data for log templates, we used publicly available data: *Blue Gene/P data from Intrepid* obtained from the computer failure data repository (CFDR) hosted by USENIX [7]. We refer to these data as the 'BlueGene' dataset. BlueGene data consist of RAS log messages collected over a period of 6 months on the Blue Gene/P Intrepid system with 11,054,588 lines. Table 3 shows examples of BlueGene log messages. As you can see from this table and Table 1, the BlueGene data have similar structures to network log messages regarding the following points: i) they are readable and one-line text messages; ii) the number of words in messages are similar; iii) words in messages have a similar tendency to become a log template (see also Section 3.2). Therefore, we determined that the BlueGene data could be used to validate our template-extraction method. In addition, each message contains `MSG_ID` that represents the types of messages such as `KERN_080B` and `CARD_0206`. Thus, we used this field as the true 'label' for the log template of each message. To quantitatively evaluate the accuracy of log templates using these labels, we chose the Rand index [27], which is a well known measure for evaluating two different clustering results. More precisely, for two arbitrarily selected messages X and Y from the data, we first set the following metrics:

- True Positive (TP): X and Y have the same `MSG_ID` and our system classifies them into the same log template.
- True Negative (TN): X and Y have different `MSG_ID`s and our system classifies them as different log templates.
- False Positive (FP): X and Y have different `MSG_ID`s and our system classifies them into the same log template.
- False Negative (TN): X and Y have the same `MSG_ID` and our system classifies them as different log templates.

Using the above notations, the Rand index is defined as

$$RAND\_INDEX = \frac{TP + TN}{TP + TN + FN + FP}.$$

From the definition, the Rand index has a value between 0 and 1, with 0 indicating that the two datasets do not agree on any pair of points and 1 indicating that the datasets are the same. In Table. 4, we show the Rand index for different E, the threshold parameter in online template extraction. The table indicates that in all cases, our template-extraction method achieved high Rand index values. Furthermore, we can see from the table that the maximum score of Rand index was 0.93190 when $E = 0.93$ and that the result worsened when E was greater than or less than 0.93. These results come from the definition of E: if E is large, then our method tends to split messages and create more template clusters; otherwise, template clusters are likely to be aggregated.

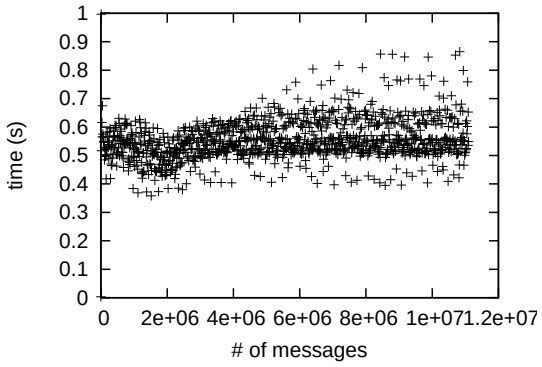


Fig. 4 Elapsed time of template extraction for every 1000 messages

Table 4 Accuracy of template extraction

Clustering threshold E	Rand index	# of log templates
0.70	0.93159	172
0.90	0.93188	432
0.93	**0.93190**	437
0.95	0.90611	524
0.99	0.90245	538
0.999	0.78480	625

Performance of incremental computation: Another evaluation metric is the performance of online template extraction. We implemented our template-extraction method by using `Python` and ran it on the experimental server with 2.0 GHz Intel Xeon CPUs and 32 GB memory. In Fig. 4, we show the elapsed time of template extraction for all 10,000 messages of actual syslog messages obtained from a certain test network. Although there were some worse cases, our method could perform roughly 20,000 messages per second on average. In a large IP network, the total volume of log messages per day is less than 100 million messages; thus, we believe that our method performs sufficiently for running on a large production network.

4.2 Proactive failure detection evaluation

Next, we explain the evaluation results regarding the proac-

tive failure-detection part of our system (presented in Subsections 3.3, 3.4, and 3.5). We used several months of log data collected from a certain working network with roughly 300 million lines for calculating the static metrics of log feature vectors. We also used 7,000 lines of maintenance procedures to obtain the 'correlation to maintenance' feature. Furthermore, to create training and test data, we selected 400 trouble-ticket datasets from different periods with the data used for static metrics. The trouble-ticket data describe when failures occurred at which host and when they recovered. We excluded the cases in which network operators did not carry out any recovery from failure (e.g. auto-recovered case) and the failures due to other reasons (e.g. service providers, or customers). We also ignored failures with no log messages since they were out of the scope of our research. To obtain labeled log chunks, we first extracted log messages in certain time windows before each failure in the trouble-ticket dataset then chose log templates that appeared before the failure. For negative datasets, we randomly cut a certain time window with no failures or maintenance and extracted log messages within that period. Finally, we obtained 350 positive and negative samples.

Evaluation metrics: For given labeled datasets, we conducted 10-fold cross validation. In all experiments, the parameters for the SVM were optimized on the basis of a pre-experiment and experience. We calculated *Recall*, *Precision*, *F1-score*, and *AUC* as evaluation matrices. *F1-score* is the harmonic mean of precision and recall and can be expressed as

$$Recall = \frac{TP}{TP + FN}, \quad Precision = \frac{TP}{TP + FP},$$

and

$$F1\text{-}score = \frac{2Recall \cdot Precision}{Recall + Precision}.$$

On the other hand, *AUC* is equal to the probability that a classifier will rank a randomly chosen positive vector higher than a randomly chosen negative one. In other words, *AUC* represents *how well feature spaces are separated* for given positive and negative instances.

Comparison across different feature selections: Figs. 5 and 6 show the *F1-score*, *AUC*, precision, and recall when selected features are varied. We adopted a template-based feature-creation method as our baseline of validation, in which we created a feature vector using *a bag-of-words* expression of log templates in a log chunk (for details, see Section 3.3). Sipos et al. [29] took a similar approach by using the message-type field in their data. In the figures, 'f' denotes frequency, 'p' periodicity, 'c' correlation with maintenance or failures, and 'b' burstiness. Furthermore, 'template' represents the template-based feature creation. We can see from the figure that by adding the features, the *AUC* score increases. This means that by using all these features, we can obtain a better feature space that can classify the data correctly. In addition, the feature space of the proposed system resulted in higher values than the template-based feature

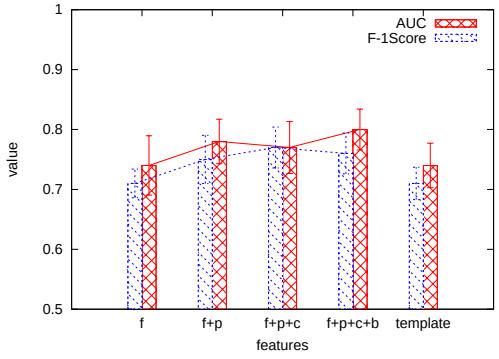


Fig. 5 $F1\text{-}Score$ and AUC results for different features selected

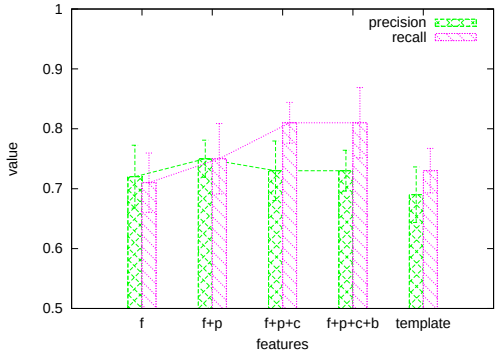


Fig. 6 Precision and recall results for different features selected

space. This result indicates that the log template is sufficient for detecting anomalies even though log templates have much richer information than keywords.

It should be noted that our feature space is not large (at most 8-dimensional feature space), as explained in Section 3.3. In addition, features (a), (b), and (d) can be calculated offline. Therefore, the proactive failure-detection part of our system does not require much computational time (less than 1 sec. for each log chunk in our experiments) although adding multiple features can increase computational time in general.

Log-generation feature space vs. keyword feature space: We next compared the feature spaces of the proposed system with the keyword-based feature spaces. To construct a keyword-based feature space, we used a *bag-of-words* expression of selected keywords. To determine the target keywords, we also defined and calculated *term frequency and inverse template frequency (tf-itf)* as the importance measure of a keyword and selected the top-M words. The *tf-itf* for a keyword w is defined as the minor modified version of the well known *tf-idf* value:

$$tf\text{-}itf(w) = \frac{1}{|T_w|} \sum_{t \in T_w} \frac{freq(w)}{freq(t)} \cdot \log\left(\frac{|T|}{|T_w|} + 1\right),$$

where T and T_w respectively represent the total set of log templates and set of log templates that include w, and $freq(\cdot)$ shows the frequency of a log template or a word. The *tf-itf* increases proportionally to the number of times a word appears in log templates but is offset by the frequency of

the word in the total set of log templates. Fig. 7 shows the comparison results of the keyword-based feature space and the feature space with keywords of the proposed system when varying the number of selected keywords. We can see that the *AUC* score of the proposed system with the feature space with keywords is slightly higher than that without keywords (see also Fig. 5). In addition, the graph shows that the *AUC* score increased as more keywords were added to the log feature vector but seemed to have an upper bound. This is partially because if the feature is large, it may over-fit to the data. On the other hand, the features of the proposed system resulted in a higher value than the keyword-based feature space for all cases. This result indicates that the abnormality of logs is determined by the logs' generation patterns rather than the keywords in messages.

Impact of log-chunk size: Another important configurable value of our system is the size of a log chunk. Since the detection timing of future failures relies on the size of a log chunk, a shorter time window is desirable for proactive operation in real time. Fig. 8 shows the *F1-score* and *AUC* results when varying the size of log chunks. Note that we used the data extracted from the same positive and negative datasets in all cases. As we can observe from the graph, *AUC* increased with the time window size. The reason for this is that if the time window is larger, more log messages are included in a chunk; thus, there is more chance for a log feature vector to absorb important log messages and their features. Although *AUC* takes the worst value when the time window is 5 min, it is more than 0.76, which is higher than the cases with keyword-based feature selection with a time window of 60 min.

4.3 Example of proactive failure detection

We give an interesting example of proactive failure detection with our system. In Fig. 9, the horizontal axis represents time, the vertical axis represents log templates (i.e., IDs of log templates), and each point represents the occurrence of the log template with its y-axis. We plotted the time series of log templates generated at a certain host in which a failure occurred on Day 8, i.e., the vertical line shows when a fault alarm was raised at this host. We found that our system could proactively detect this failure 5 days before, i.e., it could predict the future failure by using a log chunk on Day 3. More precisely, we can see that there were burst log templates before the fault alarm (IDs 4 and 5); thus, our system proactively detected the future problems. The described message on this log template shows a pair of link down and up messages, in other words, a link flap on a certain switch. Since the messages can be caused by daily maintenance and connecting new subscribers, they frequently occur throughout the network. Thus, the network operators did not monitor these messages and missed them. However, our system detected the burst occurrences of the messages since operations that cause them are all done manually, so they have less burstiness than this abnormal status. We can also see frequent and periodic log templates below the burst ones.

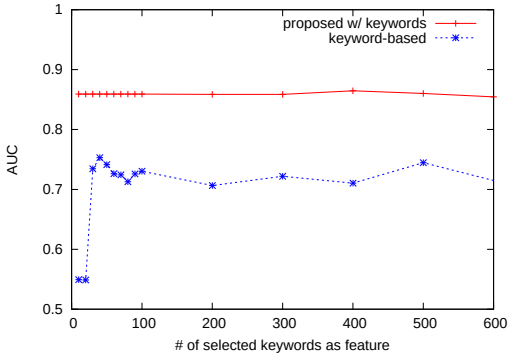


Fig. 7 Varying number of selected keywords

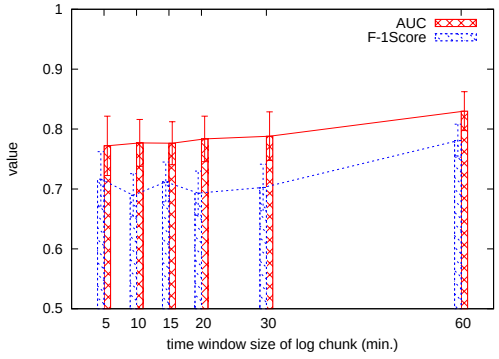


Fig. 8 Comparison with different time windows sizes

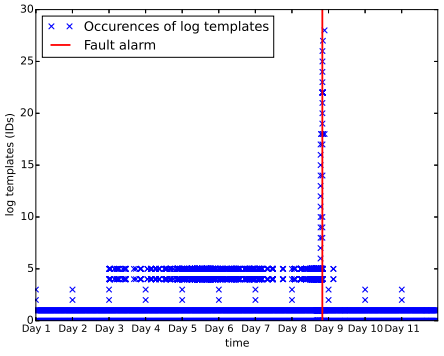


Fig. 9 Example of proactively detected failure. Vertical axis represents log template, and horizontal axis represents time. Each point corresponds to occurrence of log template, and vertical line indicates time fault alarm was raised.

These messages are associated with cron jobs and do not affect the network. As a result, our system could successfully detect the signs of a network failure from complicated and large log data. Obviously, current keyword-based monitoring and template-based monitoring systems cannot detect such an abnormal event because they are not focused on the generation patterns of log messages.

5. Conclusion

We proposed a proactive failure-detection system that ex-

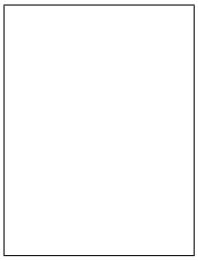
tracts the generation patterns of network log messages. To extract features from log data, we developed an online template-extraction method. We used features that characterizes the abnormality of logs on the basis of the generation patterns of logs. We found that our system can detect failures with higher accuracy than keyword-based or template-based monitoring systems.

Although our system currently learns and detects abnormal logs offline, the features and model need to be updated automatically in production networks. Thus, online proactive failure detection is our future work. Furthermore, our observation of log data has shown that there are groups of log messages because of network topology or layer dependencies (see e.g., [18]). We believe that adding such a characteristic to our features will help in improving the accuracy of proactive failure detection.

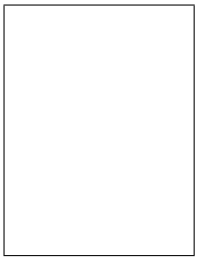
References

[1] R. Gerhards, "The syslog protocol," RFC 5424, 2009. http://tools.ietf.org/html/rfc5424.
[2] CA Spectrum. http://www.ca.com/us/root-cause-analysis.aspx.
[3] HP Software. http://www8.hp.com/us/en/software/enterprise-software.html.
[4] IBM Tivoli. http://www-01.ibm.com/software/tivoli.
[5] Logentries. http://logentries.com.
[6] Splunk. http://www.splunk.com
[7] USENIX The computer failure data repository (CFDR). http://www.usenix.org/cfdr-data.
[8] Cisco 7200 Series Routers Processor Memory Parity Errors Support Page. http://www.cisco.com/c/en/us/support/docs/routers/7200-series-routers/6345-crashes-pmpe.html.
[9] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," Proc. SIGCOMM '07, New York, USA, pp.13–24, 2007.
[10] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl, "Automating network application dependency discovery: experiences, limitations, and new solutions," Proc. OSDI '08, San Diego, USA, pp.117–130, 2008.
[11] K. Crammer, O. Dekel, J. Keshet, S. S.-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," J. Mach. Learn., vol.7, pp.551–585, 2006.
[12] C. Cortes and V. Vapnik, "Support-vector networks," Mach. Learn., vol.20, no.3, pp.273–297, 1995.
[13] Q. Fu, J.-G. Lou, Y. Wang, and J. Li. "Execution anomaly detection in distributed systems through unstructured log analysis," Proc. ICDM '09, Miami, USA, pp.149–158 , 2009.
[14] J. M. N. Gonzalez, J. A. Jimenez, J. C. D. Lopez, and H. A. Parada G. "Root cause analysis of network failures using machine learning and summarization techniques," IEEE Commun. Mag., vol.55, no.9, pp.126–131, 2017.
[15] S. Kandula, R. Chandra, and D. Katabi, "What's going on? Learning communication rules in edge networks," Proc. SIGCOMM '08, Seattle, USA, pp.87–98, 2008.
[16] S. Kandura, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed diagnosis in enterprise networks," Proc. SIGCOMM '09, Barcelona, Spain, pp.243–254, 2009.
[17] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, "Proactive failure detection learning generation patterns of large-scale network logs," Proc. CNSM '15, Barcelona, Spain, pp.8–14, 2015.
[18] T. Kimura, K. Ishibashi, T. Mori, H. Sawada, T. Toyono, K. Nishimatsu, A. Watanabe, A. Shimoda, and K. Shiomoto, "Network event extraction from log data with nonnegative tensor factorization," IEICE Trans. Commun., vol. E100-B, no. 10, pp.1865–1878, 2017.
[19] J. Kleinberg, "Bursty and hierarchical structure in streams," Proc. KDD '02, Edmonton, Canada, pp.91–101, 2002.
[20] S. Kobayashi, K. Fukuda, and H. Esaki, "Mining causes of network events in log data with causal inference," Proc. of IM '17, Lisbon, Portugal, pp.45–53, 2017.
[21] T. Li, F. Liang, S. Ma, and W. Pengo, "An integrated framework on mining logs files for computing system management," In KDD '05, Chicago, USA, pp.776–781, 2005.
[22] C. Lim, N. Singh, and S. Yajnik, "A log mining approach to failure analysis of enterprise telephony systems," Proc. DSN '08, Anchorage, USA, pp.398–403, 2008.
[23] A. Mahimkar, J. Yates, Y. Zhang, A. Shaikh, J. Wang, Z. Ge, and C. T. Ee, "Troubleshooting chronic conditions in large IP networks," Proc. CoNEXT '08, Madrid, Spain, pp.2:1–2:12, 2008.
[24] M. Mizutani, "Incremental mining of system log format," Proc. SCC '13, Santa Clara, USA, pp.595–602, 2013.
[25] R. Potharaju, N. Jain and C. N.-Rotaru, "Juggling the jigsaw: towards automated problem inference from network trouble tickets," In NSDI '13, Lombard, USA, pp.127–142, 2013.
[26] T. Qiu, Z. Ge, D. Pei, J. Wang, and J. Xu, "What happened in my network? Mining network events from router syslogs," Proc. IMC '10, Melbourne, Australia, pp.472–484, 2010.
[27] W. M. Rand, "Objective criteria for the evaluation of clustering methods," J. Am. Stat. Assoc., vol.66, no.336, pp.846–850, 1971.
[28] T. Reidemeister, M. Jiang and P. A. S. Ward, "Mining unstructured log files for recurrent fault diagnosis," Proc. IM (Mini Conf.), Dublin, Ireland, pp.377–384, 2011.
[29] R. Sipos, D. Fradkin, F. Moerchen, and Z. Wang, "Log-based predictive maintenance," Proc. SIGKDD '14, New York, USA, pp.1867–1876, 2014.
[30] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar. "Answering what-if deployment and configuration questions with wise," Proc. SIGCOMM '08, Seattle, USA, pp.99–110, 2008.
[31] T. Wang, M. Srivatsa, D. Agrawal, and L. Liu, "Learning, indexing, and diagnosing network faults," Proc. SIGKDD '09, Paris, France, pp.857–866, 2009.
[32] T. Wang, M. Srivatsa, D. Agrawal, and L. Liu, "Spatio-temporal patterns in network events," Proc. CoNEXT '10, Philadelphia, USA, pp.3:1–3:12, 2010.
[33] A. Watanabe, K. Ishibashi, T. Toyono, K. Watanabe, T. Kimura, Y. Matsuo, K. Shiomoto, and R. Kawahara, "Workflow extraction for service operation using multiple unstructured trouble tickets," IEICE Trans. Inform. Syst. vol. E101-D no. 4 pp.1030–1041, 2018.
[34] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," Proc. IPOM, Kansas City, USA, pp.119–126, 2003.
[35] R. Vaarandi and K. Podiņš, "Network IDS alert classification with frequent itemset mining and data clustering," Proc. CNSM '10, Niagara Falls, Canada, pp.451–456, 2010.
[36] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Mining console logs for large-scale system problem detection," Proc. SysML '08, San Diego, USA, pp.4–4, 2008.
[37] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," Proc. SOSP '09, Big Sky, USA, pp.117–132, 2009.
[38] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Online system problem detection by mining patterns of console logs," Proc. ICDM '09, Miami, USA, pp.588–597, 2009.
[39] K. Yamanishi and M. Maruyama. "Dynamic syslog mining for network failure monitoring," Proc. SIGKDD '05, Chicago, USA, pp.499–508, 2005.
[40] H. Yan, L. Breslau, Z. Ge, D. Massey, D. Pei, and J. Yates, "G-RCA: a generic root cause analysis platform for service quality management in large IP networks," IEEE/ACM Trans. Net., vol.20, no.6, pp.1734–1747, 2012.
[41] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis and H. Zhang, "Automated IT system failure prediction: A deep learning approach,"

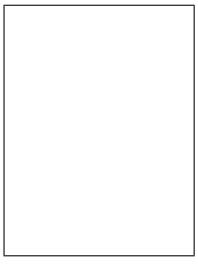
Proc. Big Data, Washington, USA, pp.1291–1300, 2016.
[42] Z. Zheng, Z. Lan, B. H. Park, and A. Geist, “System log preprocessing to improve failure prediction," Proc. DSN ’09, Lisbon, Portugal, pp.572–577, 2009.
[43] B. Zhong, Y. Wu, J. Song, A. K. Singh, H. Cam, J. Han, and X. Yan, “Towards scalable critical alert mining," Proc. SIGKDD ’14, New York, USA, pp.1057–1066, 2014.
[44] K. Q. Zhu, K. Fisher, and D. Walker, “Incremental learning of system log formats," ACM SIGOPS Oper. Syst. Rev., vol.44, no.1, pp.85–90, 2010.



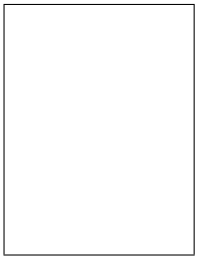
Tatsuaki Kimura Tatsuaki Kimura received the B.E. in informatics and mathematical science, the M.I. and the Ph.D. in system science, from Kyoto University, Kyoto, Japan, in 2006, 2010, and 2017, respectively. Since joining in NTT in 2010, he has been engaged in the research of management of large-scale networks and stochastic analysis of communication systems. He is currently a researcher at the NTT Network Technology Laboratories, Tokyo, Japan. He is a member of IEEE, IEICE, and the Operations Research Society of Japan (ORSJ). He received Best Paper Awards from the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM ’16) and Special Interest Group of Queueing Theory of ORSJ in 2016 and 2014, respectively, and Best Student Paper Awards from ORSJ in 2010.



Akio Watanabe Akio Watanabe received his M.E. in informatics engineering from University of Electro-Communications in 2012. He joined NTT Network Technology Laboratories, Musashino, Tokyo, Japan in 2012 as a researcher, researching network management techniques. He is a member of the IEICE.



Tsuyoshi Toyono Tsuyoshi Toyono received his B.A. and M.M.G. from Keio University in 2000 and 2002. He is currently a senior researcher at the NTT Network Technology Laboratories, Tokyo, Japan. Since he jointed NTT, he had been engaged in researches on network management, IPv6 and DNS. He joined Internet Multifeed Co. in 2009. He is a member of IPSJ, Japan.



Keisuke Ishibashi Keisuke Ishibashi received his B.S. and M.S. in mathematics from Tohoku University in 1993 and 1995, respectively, and received his Ph.D. in information science and technology from the University of Tokyo in 2005. Since joining NTT in 1995, he has been engaged in research on traffic issues in computer communication networks. He received IEICE’s Young Researcher’s Award and Information Network Research Award in 2002. He is a member of the IEEE, IEICE and the Operations Research Society of Japan.