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

Tatsuaki Kimura, Akio Watanabe, Tsuyoshi Toyono, and Keisuke Ishibashi
NTT Network Technology Laboratories, NTT Corporation,
Mushishino-shi Tokyo, 180-8585 Japan
Email: {kimura.tatsuaki, watanabe.a, toyono.tsuyoshi, ishibashi.keisuke}@lab.ntt.co.jp

***Abstract*—With the growth of services in IP networks, network operators are required to perform *proactive* operation that quickly detects the signs of critical failures and prevents future problems. Network log data, including router syslog, are rich sources for such operations. However, it has become impossible to find genuinely important logs that lead to serious problems due to the large volume and complexity of log data. We propose a log analysis system for proactive detection of failures. Our key observation is that the abnormality of logs depends on not just the keywords in the messages (e.g. `ERROR`, `FAIL`), but generation patterns such as burstiness. Our system consists of three functions: (i) extracting *log templates* automatically and quickly from a massive amount of unstructured log data; (ii) constructing *log feature vectors* to characterize the generation patterns of logs; and (iii) using a supervised machine learning approach to associate failures with the log data that appeared before them. We validated our system using real log data collected from a large network and determined its effectiveness.**

I. Introduction

Various services (e.g. IPTV, video streaming, VoIP, and internet gaming) have been deployed on recent large IP networks, in which network operators are required to perform *proactive* operations not current *reactive* operations. Service providers in fierce competition demand higher quality and reliability of the network than in previous decades. Network operators need to detect even a slight temporal event, such as a minute disconnection, because it may lead to a serious problem and service providers may suffer significant loss.

Router syslogs and alert logs generated in Network Management Systems (NMSs) are important information for current network operation. Alert messages include Simple Network Management Protocol (SNMP) trap messages Table I lists examples of logs messages. These logs contain detailed information of network elements: not only critical hardware failures but also reports on the normal status of various protocols and layers. Therefore, network log data can be considered as rich sources for performing proactive operations.

However, it has become difficult to find genuinely important log messages that are related to major network problems due to the following two reasons. First, log messages are a large number of text messages written in an unstructured format. In a large production network, the total volume of log messages may be more than tens of millions per day. In addition, since the format of logs vary by vendors or services, the types of log messages are also large in multi-vendor environment. Second, log messages are highly diverse because they contain various types of network events ranging critical hardware failures to normal login events of operators. It thus requires great experiences and previous knowledge to find meaning of log messages and make use of them.

In this paper, we propose a log analysis system for proactive failure detection. The system automatically learns the relationship between critical failures and log messages without using any previous knowledge on logs. By noticing such anomalies in advance, network operators can perform proactive operations such as preventive maintenance or arrangement of spares. For a given chunk of log messages that has appeared recently, the system automatically characterizes the generation patterns of log messages and classifies whether the current state may lead to critical failures. To use the rich features of log data, we developed an online template extraction method for our system. It can automatically and quickly transforms log messages into **log templates** by using a similarity score based on the tendency of each word to belong to a log template. The log templates are messages without parameters, such as IP addressees, and hostnames, and enables us to characterize the statistical features of log messages. Next, we create **log feature vectors** that characterize the generation patterns of log messages such as frequency, periodicity, and burstiness. Our key observation is that the abnormality of logs depends on not the keywords in log messages but these log generation patterns. For example, there are log messages that can lead to failures when they occur in sudden burst, although they do not affect the network when they occur alone. To automatically associate the log data with the critical failures in the past, we take a supervised machine learning approach by using trouble tickets as training datasets. We validated our system using a massive number of log data collected from a large production network. The experimental results show that our system can detect future network problems and achieve much better classification than current monitoring systems.

The rest of this paper is organized as follows. Section II summarizes the work related to our research. In Section III, we explain our proposed system in detail. In Section IV, we discuss several experiments that we conducted for evaluating our system. Finally, we conclude the paper in Section V.

II. Related Work

There are many commercial products for recent complicated network management and operations. NMSs, e.g., [1], [2], [3],

978-3-901882-77-7

TABLE I
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)

visualizes various metrics of network elements, such as traffic, and CPU utilization, and raise alarms based on predefined rules, e.g., keywords, severity. However, these rules often indicate apparently critical statuses and cannot capture the temporal abnormal statuses that may cause serious problems in the future. Splunk [5] is a log analysis platform, which collects, makes indexes, and visualizes logs. There are also many similar services that helps in fast analysis of data such as Logentries [4]. However, it requires great skills and domain knowledge to make efficient use of these products such as alert rule finding and log format definition.

Research of machine generated log data has increased in recent years. Yamanishi et al. [21] proposed a technique to detect system failure from server syslogs using a mixture of hidden Markov models. A system log mining method using only frequency was proposed [13]. Zheng et al. [22] introduced a log preprocessing method of filtering important logs. Xu et al. [20] analyzed console logs of large-scale hadoop systems and proposed a PCA-based anomaly detection method. A setwork IDS alerts classification method via a frequent item set mining approach was proposed in [19]. Fu et al. [10] introduced an anomaly detection technique by learning normal system behavior with a finite state automaton. The above studies focused on anomaly detection in an unsupervised manner; however, we focused on finding the signs of failures appeared in the past. Thus, we took a supervised machine learning method and associated failures described in trouble tickets and log messages occurred before them. SyslogDigest [14] targets the router syslogs in a Tier 1 scale network; however, it just constructs digest information by grouping log messages within relevant routers. Kimura et al. [11] proposed a modeling and event extraction method of network log data using a tensor factorization approach; however, they did not focus on detection of anomalies. The most closely related work is that by Sipos et al. [17]. They presented a multiple-instance learning approach for predicting equipment failures by analyzing system log data. In contrast with our work, they did not use the generation patterns of log data and automatic template extraction. Similarly, Reidemeister et al. [16] also did not use these features. They studied recurrent failure detection from unstructured log based on a decision tree classifier.

III. PROPOSED SYSTEM

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

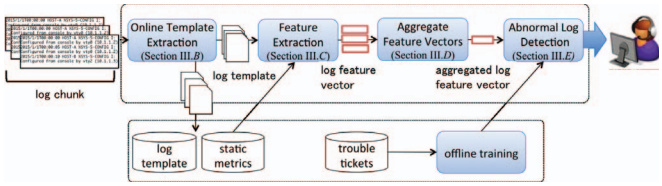

Fig. 1. System overview

A. System Overview

The main objective with our system is to automatically learn the relationship between critical failures and log data and to proactively detect an abnormal pattern of log messages that leads to future problems by analyzing recent logs. Fig. 1 gives an overview of our system. It first splits log messages by a certain time window (e.g. 15, 30 min.) and groups them into chunks with the same hosts, called **log chunks**. For a given log chunk, our system classifies whether current status may lead to critical problems and reports it to network operators. To achieve our goal, the system first preprocesses unstructured logs and transforms each message into a log template that enables us to characterize the log generation patterns (Section III-B). It then creates log feature vectors that characterize the generation patterns of log messages for each log template in a log chunk (Section III-C), and the extracted feature vectors are aggregated within a chunk (Section III-D). Finally, machine learning classifier detects future failures for the aggregated log feature vector (Section III-E). The classifier model is trained offline using network trouble ticket data. We now give detailed explanation for each component.

B. Online Log Template Extraction

Network logs include various types of messages ranging from critical failure to normal console logs. We can see from Table I that there is no unified rule for description of log messages. Since messages with unique error IDs or process IDs may never appear twice, it is unrealistic to statistically analyze raw messages to extract features.Therefore, we need to focus on not log messages but log templates, in which error IDs or process IDs are removed. We give an example of log template 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.

Template extraction methods have been proposed e.g. [18], [14], [11]; however, they are all offline batch schemes. The

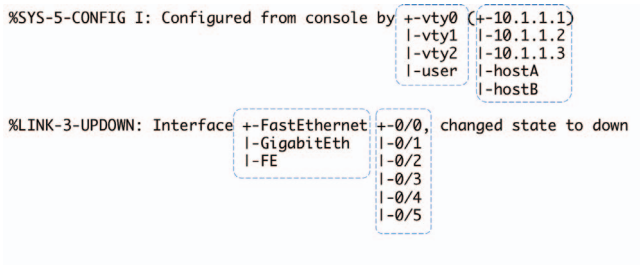

Fig. 2. Examples of template clusters. Words in box represent parameter words, and newly arriving one will be stacked in same positions.

TABLE II
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	<, >, =, :

format of messages may dynamically change in the future; in addition, we need to observe for a long period to capture all templates. Therefore, we developed an online template extraction method, which can learn templates in an incremental manner. The main ideas of the method are: (i) classification of each word based on the tendency to belong to a log template; and (ii) online clustering of arriving messages by regarding a log template as a cluster of messages and by using **log similarity** between template clusters and messages based on the classes of words. We explain the key features of our method step by step below.

(a) **Classification of words** From the observation of log messages, symbol words, such as "=" or ":", are likely to belong to log templates; and on the other hand, numerical words, such as process IDs, can be considered as parameters. According to this idea, we first classify the words in the sense of tendency to belong to a log template. The detailed definition of classification of words is described in Table II. 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. According to the definition, the value of $\boldsymbol{w}$ is typically set as $w_1 \leq w_2 \leq \cdots \leq w_5$.

(b) **Online message clustering** Next, for each arriving log message, we perform online clustering so that the message is assigned to the cluster with the highest similarity. To do this, we define the following log similarity between a cluster C and a 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 appeared in both C and X. 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 $\mathcal{C} = \emptyset$;
2: **for** each message X **do**
3: $GetWordClass(X)$;
$C := FindHighestLogSimilarity(\mathcal{C}, X)$;
4: **if** $LogSimilarity(C, X) \geq E$ **then**
5: append X to cluster C;
6: **else**
7: create a new cluster from X;
$\mathcal{C} := \mathcal{C} \cup \{X\}$;
8: **end if**
9: **end for**

Fig. 3. Online template extraction pseudo-code

(c) **Parameter optimization** To obtain the most efficient result from our method, we need to optimize the weight parameter $\boldsymbol{w}$ and E. 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 feed-backing 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 IV, we used PA-I [8] as the learning algorithm, which is a well-known online supervised classifier. Roughly speaking, it makes a minimum change to its weight vector when its prediction is wrong.

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 the template cluster that achieves the highest $LogSimilarity$ with X. If the value 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.

C. Feature Extraction

After template extraction, the system attempts to capture the features of log templates in each log chunk to characterize the generation patterns. Suppose that templates in a log chunk at a host h are $\{t_1, \ldots, t_N\}$. We then create a log feature vector $\boldsymbol{x}_i$ $(i = 1, \ldots, N)$ for each log template t_i. The simplest way to construct a log feature vector is a *bag-of-words* expression of words used in natural language processing area. 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 to a keyword-based monitoring such that seemingly abnormal words, e.g. `ERROR`, `DOWN`, are not always related to problems. For example, a user session disconnection event causes log templates; however, this occurs throughout the network on a daily basis. Therefore, we focus on not words in log messages but the generation patterns of log messages such as frequency, periodicity, or burstiness. For instance, periodic log messages, e.g., those induced by cron jobs and SNMP polling, can be considered

as normal. According to these observations, we select the following features from log generation patterns:

(1) Frequency: 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 as normal. On the other hand, operators need to check infrequent messages.

(2) Periodicity: There are periodic log messages, such as messages induced by cron jobs, Internet Control Message Protocol and SNMP polling, and daily maintenance operations. Although these periodic log message may be infrequent, they are unlikely to be related to failures. Since a wide range of granularity for the period can be considered, we define the periodicity of log templates as the coefficient of variation of the observed number of templates within each *hour, day*, and *week*. Formally, for each log template t, let

$$Periodicity(t, \mathrm{ITVL}) = \sqrt{\sigma^2_{t,\mathrm{ITVL}}} \Big/ \overline{D}_{t,\mathrm{ITVL}},$$

where $\mathrm{ITVL} \in \{hour, day, week\}$ represents an interval and $\overline{D}_{t,\mathrm{ITVL}}$ and $\sigma_{t,\mathrm{ITVL}}$ are the mean and standard deviation of the observed number of ts within the intervals. Although there are many candidates for calculating periodicity, such as Fourier transform, the reason we choose this simple metric is that manual daily operation is not strictly periodic (e.g. some are done in the morning and some are done in the evening).

(3) Burstiness: Some log messages become failures when they occur in sudden burst, 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. However, if the bit error occurs more frequently than before, the module has the potential to crash and should be replaced (see e.g. Cisco's support page [7]). Therefore, burstiness of log messages is an important feature for discovering the signs of future failures.

To calculate bursty features, we simply adopt Kleinberg's burst detection algorithm [12]. Briefly, this method 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. The state with a higher Poisson parameter indicates the burst state. Thus, we use this 'burst level' as a bursty feature. Note that we apply Kleinberg's algorithm to the log templates in a log chunk combined with *dummy log messages*, which are artificially generated using the mean value of the interval time of each log template. By adding this, the algorithm can detect the base line of interval time.

(4) 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 is caused by maintenance. To take into account the tendency to appear during maintenance, we add the following feature:

$$\frac{(\#\text{of observed templates during maintenance})}{(\#\text{of template observed in a whole period})}.$$

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

We calculate the above features (1)–(4) for each log template $\{t_i; i = 1, \ldots, N\}$. Features (1) and (2) are also counted for each tuple (host, log template), i.e., $\{(h, t_i)\}$, 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.

D. Feature Aggregation

Next, our system aggregates log feature vectors and obtains a single vector that fully characterizes the log chunk. After feature extraction, we obtain the set of log feature vectors $\{\boldsymbol{x}_{t_1}, \ldots, \boldsymbol{x}_{t_N}\}$ 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 $\{\boldsymbol{x}_{t_i}\}$. 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: More specifically, our system calculate the mean values for (1) frequency, (2) periodicity, and (4) correlation to failures; the max values for (3) burstiness; and the minimum values for (4) correlation to maintenance features, respectively.

E. Machine-Learning-Based Proactive Failure Detection

As a final component 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 technique using network trouble ticket data. More precisely, we consider a binary classification problem in which given h-th failure label $y_h \in \{1, -1\}$ (1 represents failure state, called *positive* and -1 is normal state, called *negative*) and corresponding feature vectors $\boldsymbol{x}_h$, we classify an unlabeled feature vector into binaries. In our research, we adopt a support vector machine (SVM) with the Gaussian kernel [9] for supervised machine learning. An SVM is a well-known powerful tool for binary classification. It constructs a hyperplane that maximizes the margin between two classes corresponding to the labels. By using the kernel, the input feature space is mapped into a certain high dimensional space by non linear transformation; and thus, the model can learn linearly non-separable dataset.

IV. Experiments

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

A. Online template extraction evaluation

We first explain the evaluation results for the online template extraction part presented in Section III-B. Due to the lack of ground truth data for log templates, we used the data in another domain: *Blue Gene/P data from Intrepid* obtained from the computer failure data repository (CFDR) hosted

TABLE III
ACCURACY OF TEMPLATE EXTRACTION

clustering threshold E	Rand index	# of 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

by USENIX [6]. We refer to this as the 'BlueGene' data. 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. Each message contains `MSG_ID` that represents the types of messages, e.g., `KERN_080B` and `CARD_0206`. Thus, we used this field as the true 'label' for the log template of each message. To quantitatively evaluate accuracy of log templates, we chose the Rand index [15], which is a well-known measure for evaluating two different clustering results. More precisely, for two arbitrary selected messages X and Y from the data, we first set the following:

• True Positive (TP): X and Y have the same `MSG_ID` and our system classifies them into the same template.
• True Negative (TN): X and Y have different `MSG_IDs` and our system classifies them as different templates.
• False Positive (FP): X and Y have different `MSG_IDs` and our system classifies them into the same template.
• False Negative (TN): X and Y have the same `MSG_ID` and our system classifies them as different 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 datatests do not agree on any pair of points and 1 indicating that the datasets are exactly the same. In other words, the Rand index can be considered as *an accuracy of clustering*. In Table. III, we show the Rand index for different E. The table indicates that in all cases, our template extraction method achieve 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 when E is greater than or less than 0.93, the result worsened. 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[1].

B. Proactive failure detection evaluation

Next, we explain the evaluation results from the proactive failure detection part of our system (presented in Subsections III-C, III-D, and III-E). We used several months of log data captured from a certain working network with roughly 300 million lines. 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 data sets. The trouble ticket data describes when failures occur at which host and when they recover. We excluded the cases in which network operators did not perform any recovery from failure (e.g. auto-recovered case). We also ignored the case when the templates that occurred both before and after the failures are the same, because they were the out of the scope of our research. To obtain labeled log chunks, we extracted log messages in certain time windows before each failure in the trouble ticket data set. For negative data sets, we randomly cut 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: To compare the performance of proactive detection system with different parameter settings, we calculated AUC as an evaluation metrics. 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, it represents *how well feature spaces are separated* for given positive and negative instances. We also calculated $Recall$, $Precision$, and $F1\text{-}score$ to evaluate the accuracy of the classification. Note that $F1\text{-}score$ is the harmonic mean of precision and recall, i.e.,

$$Recall = \frac{TP}{TP + FN}, \quad Precision = \frac{TP}{TP + FP},$$
$$F1\text{-}score = \frac{2Recall \cdot Precision}{Recall + Precision}.$$

For given labeled data sets, we conducted 10-fold cross validation.

Comparison across different feature selections: Fig. 4 and 5 show the $F1\text{-}score$, AUC, precision, and recall when selected features are varied. We adopted template-based feature creation method as our baseline of validation. Sipos et al. [17] took the similar approach to this feature creation, in which we created feature vectors using *a bag-of-words* expression (see Section III-C) of log templates in a log chunk, while they used *message types* described in their data. In the figures, 'f' denotes frequency, 'p' denotes periodicity, 'c' denotes correlation with maintenance or failures, and 'b' denotes burstiness. Furthermore, 'template' represents the template-based feature creation. We can see from the figure that the feature space of the proposed system achieved higher values than the template-based feature space. The result indicates that the bag of words expression is insufficient for detecting the anomalies, although log templates have much richer information than keywords. Finally, from Fig. 5, it can be said that the proposed system improved succeeded in improving both precision ($+5.8\%$) and recall ($+9.5\%$) simultaneously.

Log generation feature space vs. keyword feature space: We next compare the feature spaces of the proposed method with the keyword-based feature spaces. To construct a

[1]The performance of our algorithm also depends on $\boldsymbol{w}$ determined by a supervised classifier (see Section III-B). In this experiment, we chose $\boldsymbol{w}$ as $[0.1, 0.2, 1.0, 2.0, 3.0]$. We conducted several experiments with different $\boldsymbol{w}$ and fixed E, however, we found that E has more impact than varying $\boldsymbol{w}$. Thus, we do not discuss in detail here due to space limitations.

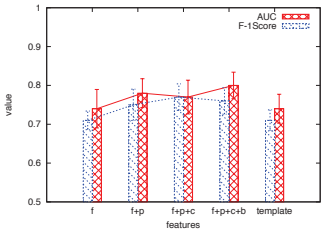

Fig. 4. F1-Scores and AUC results for different features selected

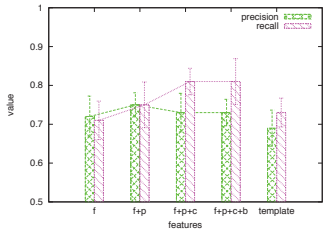

Fig. 5. Precision and recall results for different features selected

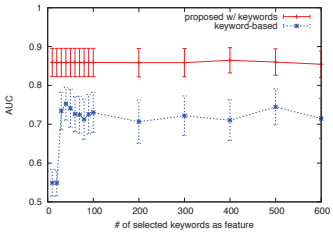

Fig. 6. Varying number of selected keywords

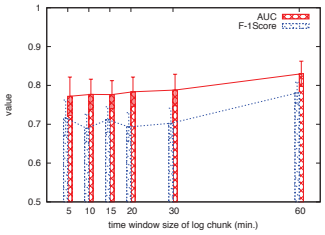

Fig. 7. Comparison with different time window sizes

keyword-based feature space, we used a *bag-of-words* expression of selected keywords. To do this, we calculate *term frequency and inverse template frequency (tf-itf)* as the importance measure of a keyword, and select top-M words. The *tf-itf* for a keyword w is defined as the minor modified version of the well-known *tf-idf*:

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

where T and T_w represent the total set of templates and set of templates that include w, and $freq(\cdot)$ shows the frequency of a log template or a word. The *tf-itf* value increases proportionally to the number of times a word appears in templates, but is offset by the frequency of the word in the total set of log templates. Fig. 6 shows the results of the proposed feature space against keyword-based feature space when varying the number of selected keywords. The graph shows that the features of the proposed method achieve a higher value than all cases. This result indicates that the abnormality of logs is determined by their generation patterns, rather than the keywords in messages.

Impact of size of log chunk: Finally, we investigated the impact of the size of a log chunk. Since the detection timing of future failures relies on it, a shorter time window is desirable. Fig. 7 shows the $F1\text{-}score$ and AUC results when varying the size of log chunks. As we can observe from the graph, AUC increased with the time window size. The reason for this is that if the size of time window is larger, more log messages are included in a chunk; thus, there would be more chance for a log feature vector to absorb important features. Although AUC takes the worst value when the time window size is 5 min., it is higher than the cases with keyword-based feature selection with time window size of 60 min.

C. Example of proactive failure detection

We give an interesting example our system detected. In Fig. 8, we plot the time series of log templates corresponding to the detected host. The vertical line shows when a fault alarm was raised. The figure shows that there are burst log templates before the fault alarm; and thus, our system proactively detected the future problems. The described message on this log templates show a pair of link down and up messages, i.e., a link flap. Since the messages can be caused by daily maintenance or connecting new subscribers, the network

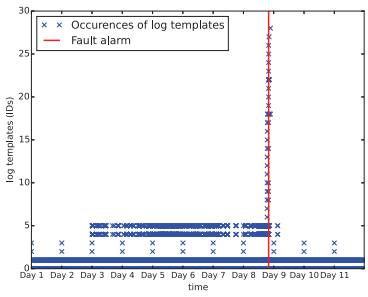

Fig. 8. Example of proactively detected failure. Vertical axis represents log template and horizontal axis represents time. Each point corresponds to occurrence of log template and vertical line indicates time fault alarm were raised.

operators did not monitor this messages and missed them. On the other hand, our system detected them because operations that causes them are all done manually; and thus they have less burstiness. In addition, we can see frequent log templates and periodic ones below the burst ones. These messages are associated with cron jobs and do not affect the network. As a result, our system could successfully detect the sign of a network failure, which cannot be detected by current keyword-based or template-based monitoring.

V. Conclusion

We proposed a proactive failure detection system from the generation patterns of network log messages. To extract features from log data, we developed an online template extraction method. We also developed a future extraction method that characterizes the abnormality of logs based on the generation patterns of logs. We confirmed that our system can detect failures with higher accuracy than keyword-based or template-based monitoring.

Although our system currently learns and detects abnormal logs in offline, automatic update of the features and the model is important in production networks. Thus, online proactive failure detection is our future work. Furthermore, from the observation to log data, there are groups of log messages because of network topology or layer dependencies (see [11]). We believe that adding such a characteristic to our features helps in improving the accuracy of future failure detection.

REFERENCES

[1] CA Spectrum. http://www.ca.com/us/root-cause-analysis.aspx.

[2] HP Software. http://www8.hp.com/us/en/software/enterprise-software.html.

[3] IBM Tivoli. http://www-01.ibm.com/software/tivoli.

[4] Logentries. http://logentries.com.

[5] Splunk. http://www.splunk.com.

[6] USENIX The computer failure data repository (CFDR). http://www.usenix.org/cfdr-data.

[7] 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

[8] K. Crammer, O. Dekel, J. Keshet, S. S.-Shwartz, and Y. Singer, Online Passive-Aggressive Algorithms, In *Proc. NIPS*, 2003.

[9] C. Cortes and V. Vapnik, Support-vector Networks, *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[10] Q. Fu, J.-G. Lou, Y. Wang, and J. Li. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis, In *Proc. ICDM*, 2009.

[11] T. Kimura, K. Ishibashi, T. Mori, H. Sawada, T. Toyono, K. Nishimatsu, A. Watanabe, A. Shimoda, and K. Shiomoto, Spatio-temporal Factorization of Log Data for Understanding Large-scale Network Events, In *Proc. INFOCOM*, 2014.

[12] J. Kleinberg, Bursty and Hierarchical Structure in Streams, In *Proc. KDD*, 2002.

[13] C. Lim, N. Singh, and S. Yajnik, A Log Mining Approach to Failure Analysis of Enterprise Telephony Systems, In *Proc. DSN*, 2008.

[14] T. Qiu, Z. Ge, D. Pei, J. Wang, and J. Xu, What Happened in my Network? Mining Network Events from Router Syslogs, In *Proc. IMC*, 2010.

[15] W. M. Rand, Objective Criteria for the Evaluation of Clustering Methods, *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[16] T. Reidemeister, M. Jiang and P. A. S. Ward, Mining Unstructured Log Files for Recurrent Fault Diagnosis, In *Proc. IM (Mini Conf.)*, 2011.

[17] R. Sipos, D. Fradkin, F. Moerchen, and Z. Wang, Log-based Predictive Maintenance, In *Proc. SIGKDD*, 2014.

[18] R. Vaarandi, A Data Clustering Algorithm for Mining Patterns from Event Logs, In *Proc. IPOM*, 2003.

[19] R. Vaarandi and K. Podiņš, Network IDS Alert Classification with Frequent Itemset Mining and Data Clustering, in *Proc. CNSM*, 2010.

[20] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, Detecting Large-scale System Problems by Mining Console Logs, In *Proc. SOSP*, 2009.

[21] K. Yamanishi and M. Maruyama. Dynamic Syslog Mining for Network Failure Monitoring, In *Proc. KDD*, 2005.

[22] Z. Zheng, Z. Lan, B. H. Park, and A. Geist, System Log Pre-processing to Improve Failure Prediction, In *Proc. DSN*, 2009.