Department of Information and Communication Engineering
Graduate School of Information Science and Technology
THE UNIVERSITY OF TOKYO

Master Thesis

Network System Anomaly Detection by Log Analysis

(ログ解析によるネットワークシステム異常検知)

Kazuki Otomo
大友 一樹

Supervisor: Professor Hiroshi Esaki

February 2019

Abstract

System logs are useful to understand the status of and detect faults in large scale networks. However, due to the diversity and volume of these logs, log analysis requires much time and effort. In addition, two issues must be considered in automated log analysis that have not been solved: (1) a gap between statistical anomaly and real problem, (2) adaptivity to network system change. This work tackles on these two issues, using real log data obtained from SINTE4, a nation-wide network in Japan. First, we focus on burstiness and causality of log time series data and analyze the usefulness of detecting log bursts. Through this analysis, we find metrics to detect meaningful burst for troubleshooting. Next, we propose a latent variable based log anomaly detection method, which works without data-specific pre-processing and pre-defined anomaly, and thus it is adaptive to the change of trend. Through evaluation, we demonstrate that our proposed method achieves 14.5% higher recall and 3% higher precision than a traditional PCA based method. We confirm findings in this work ease troubleshooting of network system faults with several case studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Network reliability is a crucial concern in the Internet. For example, for ISP network operation, high availability and fault tolerance have been critical requirements while its network system is getting more complicated and large-scaled. Therefore, network operators are required to troubleshoot network problems as fast as possible. To troubleshoot network problems, network operators investigate network behavior relying on data obtained from network system. For example, traffic data, resource usage data gathered by SNMP [1], network logs and so on. Analyzing network logs is one of the most useful methods to understand natures of networks. Unlike other data, network logs are familiar format for network operators because logs are text data. By reading log messages, network operators can understand when and where and what happened in the network system. In operational networks, syslog [2] is widely used protocol for collecting network logs and it allows us to gather logs from all devices at one server. With these logs, the operators investigate detailed status and events for each devices in the network system.

However, due to rapid increase of the scale and complexity of network system these days, the amount of logs is too huge to manually analyze. In addition, since formats of log messages are defined by each device vender, software, OS and so on, the representation of log messages is completely different and thus the number of kind of log is also huge large. Therefore, automatic log analysis methods are highly required.

Many log analysis methods for finding anomalies and their root causes have been proposed to realize automatic log analysis [3, 4, 5, 6, 7, 8]. In many cases, one first classifies the logs by their message type (i.e., *log template*) then treats them as statistical time series to be later processed through statistical analysis. System logs have an intrinsic nature that makes automatic analysis very difficult; their time series show a variety of characteristics such as periodicity, burstiness, randomness, and sparseness, compared with other time series data. Most studies have been devoted to developing effective pre-processing methods (e.g., filtering, smoothing, denoising) and appropriate discriminative log time series features for analyzing their own data. In other words, these existing methods highly depend on each dataset. Thus, we require enough domain knowledge about the underlying networks for optimizing the analysis methods in order to use these methods in other system.

After analyzing logs by existing methods, we find anomalies or root causes in them. However, anomalies and root causes are statistical results and not always related to real serious problems. For example, network device updates may cause infrequent behavior, which are statistically outliers but not real serious problems. To utilize the analysis results, we have to consider severity or importance of them because these detected anomalies are not always system anomaly but just statistical outliers.

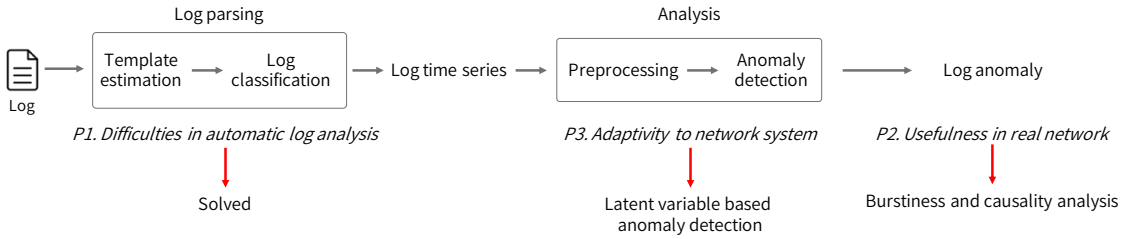

Fig. 1.1: Analysis overview

1.2 Problem statement

Figure 1.1 shows general processing flow of log anomaly detection and each problem comes from each step. In the figure, we can see the difficulties mentioned in the previous section as following three problems in automatic log analysis.

■1. Gap between numerical data and text data Since log messages are free format text data, it is difficult to directly analyze logs. Thus, we first have to convert log messages into numerical data. As a preprocessing, almost all past literatures estimate formats of log messages from raw log messages and categorize them by their formats.

■2. Gap between statistical anomaly and real problem By using log template and log time series, we can apply existing statistical analysis methods to log messages. Then, we find statistical characteristics, anomalies, or outliers by these statistical analyses. However, these statistical outliers are not always important in real problems. In other words, these characteristics are not always useful knowledge for network operation. Therefore, we must evaluate the importance of analysis results in a real network system.

■3. Adaptivity to the change of network system behavior Network system behavior depends on network topology or policies. They are different for each network system and thus network system behavior is totally different for each system. Even in the same network, they are changing in time. Thus, the analysis method should be available in various network systems. Focusing on the anomaly detection in network system, the method must be capable to detect unknown pattern of attacks or anomalies since the way of attacks always change.

In many literatures, processing free format text data, which is the first problems shown as P1. in Fig. 1.1, is realized by log template estimation for converting text data into numerical data. They also rely on preprocessing to mitigate difficulty of statistically handling logs, such as sparseness, long-tailed distribution. These two processes are sophisticatedly tuned. Thus existing methods work very well against known anomalies in the specific network system.

In this study, we tackle on the second and third problems. First, we focus on log burstiness, which is one of the most typical time series anomalies and analyze how useful observed bursts in log messages for network operation. We conduct this analysis for the second problem. Next, towards more general approach, we propose latent variable based log anomaly detection for the third problem.

1.3 Contributions

The contribution of this work is as follows. First, in order to evaluate the usefulness of log anomalies, we combine two approaches. One is a log causal inference, which is knowledge mining approach from log data, and the other is burst detection, which is naive approach to detect anomaly

in time series data. We conduct three types of analysis with causal inference results and extract meaningful burst data for troubleshooting from all the detected bursts. Second, we propose our latent variable based method for highly diverse log time series data without any data-specific preprocessing. We discuss the effectiveness of the method when we applied it to real network syslog data. Our method outperforms traditional PCA based anomaly detection with 14.5% higher recall and 3% higher precision. Our case study shows useful results which are not detected by traditional PCA based method. Through these two works, we partially solve the second problem as shown in §1.2 in our burstiness and causality analysis and we confirm our latent variable based method can handle the third problem.

1.4 Structure of the thesis

In chapter 2, we introduce related work and clarify the problems in system log analysis. In chapter 3, we explain the SINET4 dataset, which is used in evaluations in this work. Next, we conduct burstiness and causality analysis in chapter 4 and latent variable based anomaly detection in chapter 5. Finally, we discuss the whole results in chapter 6 and conclude our work in chapter 7.

Chapter 2

Related work

2.1 System log analysis

Many prior literature tackle on automated system log analyses. The automated system log analysis helps us detect system failures, identify the root cause of them, and predict the failures from tons of log messages. He et al. [9] discuss common functionality of the automated system log analysis by comparing prior literatures: (1) log collection, (2) log parsing, (3) feature extraction, and (4) anomaly detection, root cause identification, or failure prediction. We conduct our analysis along with this classification. We need (1)-(3) processes to perform numerical or statistic methods to log messages because system logs are string data. In process (4), many analysis algorithms have been proposed.

There are other approaches which focus on mining knowledge from logs. Their goal is not anomaly detection or prediction but discover hidden relationship in log data for providing useful information for network operators. They also need (1) - (2) process since they also rely on statistical methods. In this chapter, we introduce prior literatures for each process: (2) log parsing, (4) anomaly detection and knowledge mining approach.

2.2 Log parsing

Generally, log messages are text data. Thus, we have to estimate log format and classify them based on their format in order to apply statistical analysis method to log data. In this section, we introduce log template (i.e., format of log) estimation methods.

Basically, the goal of log template estimation is to identify variable words and constant words in log messages. Figure 3.1 shows an example of translation from raw log messages into a log template. As shown in this figure, above two log messages are different but have the same words in the same position. We call these same words as constant words, and different ones as variable words. Then, we regard variable words as wild cards and compose log templates using constant words and variable words as shown in the bottom of Figure 3.1. In order to determine whether a word is constant or variable, previous literatures focus on the occurrence position and frequency of the word in a log message. Vaarandi et al. [10] proposed a simple method focusing on the word occurrence position and frequency. They first count the number of word occurrences along with the location of word occurrence. If a word occurs more than a threshold at the same position, they consider the word is constant. This method is called SLCT (Simple Logfile Clustering Tool), that is used in [11]. Makanju et al. [12] introduced IPLoM (Iterative Partitioning Log Mining). This method generate log templates by three step. They basically rely on the word occurrence position and frequency but they also focus on length of log messages, word occurrence position tendency and co-occurrence of words in a log message. This method is used in [13, 14].

Since these two methods are largely depending on the amount of log appearances, it is difficult to estimate log templates which is low occurrence frequency. However, we need to estimate log

templates of rarely appeared logs because the occurrence frequency of anomalous logs is lower than normal ones. Thus, there are approaches which overcome this problem of log occurrence dependency.

Mizutani et al. [15] proposed SHISO (Scalable Handler for Incremental System lOg) which is a log template estimation and incremental log clustering tool with a tree structure. This method does not depend on log occurrence frequency but similarity between log messages. Thus, they can estimate even rarely appeared log templates. He et al. [16] also proposed Drain which is tree-structure-based log parsing approach. This approach achieves higher accuracy and faster processing time than IPLoM and SHISO with benchmark dataset.

He et al. [9] compared four log template estimation methods. They selected SLCT, IPLoM (both mentioned above), LKE (Log Key Extraction) [17] and LogSig [18] and evaluated them with log dataset for benchmark. As a result, the best method that achieved the highest accuracy of log template estimation is different for each dataset. Thus, we need to carefully select log template estimation method or optimize existing method for our own dataset.

In this work, we employ the log template estimation method proposed by Kobayashi et al. [19]. Details are written in §3.

2.3 Log anomaly detection

To detect system anomaly with system logs, there are a lot of existing methods [5, 6, 20]. In recent work, Baseman et al. [3] employ a graph analysis with a relational learning and kernel density estimation, and generate clusters of related syslog messages. Zhong et al. [4] proposed an anomaly detection method for both device and network errors with fine log time series feature creation. They successfully extract anomalous behavior from super computer syslog data with low false positives. To diagnose root causes of system failure [21, 14] and predict system failure [4, 6] are also studied well. Lu et al. [7] propose a root cause analysis method with log messages and resource logs (e.g. CPU and memory logs). They focus on the task execution time and define features. To predict system failure, Kimura et al. [22] propose a network fault prediction system based on system logs. They analyze log features related to system faults in advance. With trouble ticket data, they incrementally analyze the log features and compare with the system fault log features.

Overviewing these existing methods, we find there are two issue to be carefully considered (also mentioned at §1). First one is that log anomaly do not directly indicate system anomaly. Many existing methods try to solve this issue with combining domain knowledge or sophisticated feature extraction. The goal of our work in §4 is to extract meaningful information for troubleshooting from system logs and our strategy is contextual log analysis with causal inference. The causal relationship of log anomaly relates to the real system anomaly although the simple co-occurrence of log anomaly includes a large number of false positives. Our method provides causality between groups of logs related to burstiness and the results are useful information for troubleshooting.

Second one is adaptivity in other network system and change of network status. As these existing methods are based on data specific feature creation, they perform well in each considered environment. However, to apply these methods to other network systems, we have to re-define features to optimize these methods. This process requires deep domain knowledge of the underlying network systems to make efficient use of these methods. In addition, these methods miss unknown or new anomalies, which are not captured by the pre-defined features. Instead, in latent variable based anomaly detection, we focus on an approach that does not require pre-definition of anomalies but learns the normal state of the system from log data.

Recently, some studies have applied deep learning techniques to anomaly detection in logs [23, 24, 25, 26]. Du et al. [27] proposed a log anomaly detection method by using Long-Short Term Memory (LSTM). They built LSTM models for each type of logs and whole log time series in

application level logs (OpenStack and HDFS logs). However, we have to handle network logs which are larger amount and more diverse than application level logs. Therefore, we circumvent anomaly detection directly from dataset but try to reduce the dimension of input dataset by latent variable analysis.

We focus on anomaly detection without specific feature creation for more general analysis of logs. PCA is a standard algorithm of learning or obtaining time series features from data. Lakhina et al. [28] proposed a general traffic analysis method based on feature learning with PCA. They successfully detected traffic anomalies without specific definitions of those anomalies. However, as log data are highly sparse and discrete, which differ with traffic data, it is not enough to learn features with such a naive approach (i.e., PCA). Thus, we use CVAE, which is based on a higher assumption that observed data are subject to latent variables. We choose a PCA based method as a baseline algorithm and aim to outperform it with our proposed method.

2.4 Knowledge mining from log data

There are methods of giving new insights for operators with knowledge mining from log data [29, 30]. Hacker et al. [31] introduced a log classification method based on the severity of network operation. These methods do not require predefined features because the features characterizing the data can be obtained through learning. Note that they do not aim at anomaly detection but knowledge mining.

Kobayashi et al. [8] proposed a time series causal inference method in network logs. Causal inference is a statistical technique to identify a causal relationship between events. A well-known causal inference algorithm is PC algorithm [32, 33] that is based on conditional independence. The conditional independence distinguishes causal two events from co-occurred two events. Chen et al. [34] apply the PC algorithm to a set of time series (e.g., RTT, TCP window size) for identifying the source of network traffic delay.

The PC algorithm, however, has an issue for applying to system logs. Appearance of log messages is discrete and sparse compared to other metrics such as CPU or memory usage. It makes the causal analysis difficult. To overcome this issue, Kobayashi et al. [35] remove normal logs such as periodically or constantly appeared logs, and then apply the PC algorithm to an event time series to extract causal relationships. They apply their proposed algorithm to 15 months long syslog messages and successfully extract a small number of meaningful network events with causal relations. In this work, we use the same dataset and the causal inference results. We refer to a set of these results as the causality DB in §4.

Chapter 3

Dataset and preprocessing

In this work, we use a large scale syslog dataset obtained from SINET4 [36] and a template estimation method. SINET is a nation-wide research and education network in Japan. In this chapter, we mention about the detail of SINET4 dataset and log template estimation method.

3.1 SINET4 dataset

We use a set of network logs collected at SINET [36], a Japanese research and education network. This network connects over 800 academic organizations in Japan and consists of eight core routers, 50 edge routers, and 100 layer-2 switches. Figure 3.2 shows layer-2 topology of SINET4. As shown in the figure, SINET4 is a tree topology that consists of eight areas.

Table 3.1: SINET4 Dataset

logs	devices	templates	term
34.7M	130	1789	456 days (2012/01/01 - 2013/03/31)

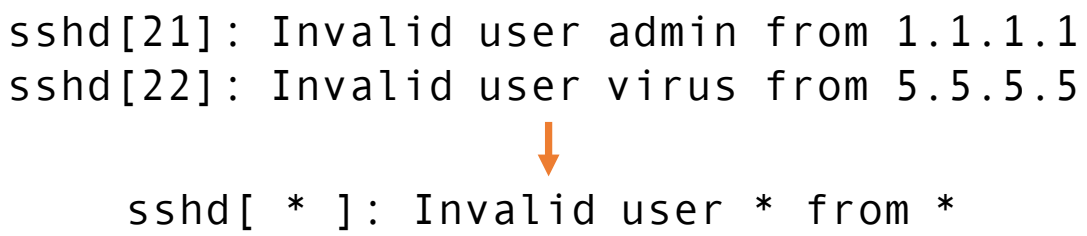

Fig. 3.1: Example of log templates

3.2 Log template generation

As log messages are string data, statistical approaches cannot be applied directly. Thus, we generate log templates from raw log messages and then we classify logs into log templates and extract time-series data from their time stamps for each template per device. The log template is a format of log messages composed of variables (e.g., IP address, port number) and other constants. Template generation problem is a well-known problem in log mining [15, 10]. In this work, we adopt a supervised learning approach proposed by Kobayashi et al. [19]. This algorithm is based on a CRF (conditional random field), which is well-studied in natural language processing and generates log templates composed by description words and variable words from original log

messages. An example of log templates and raw log messages are shown in Fig. 3.1. The top two lines are raw log messages and the bottom line is a corresponding log template generated from them. “*" represents variable words. We manually fixed misclassified templates.

As the result of this processing, we generated 1,789 unique log templates.

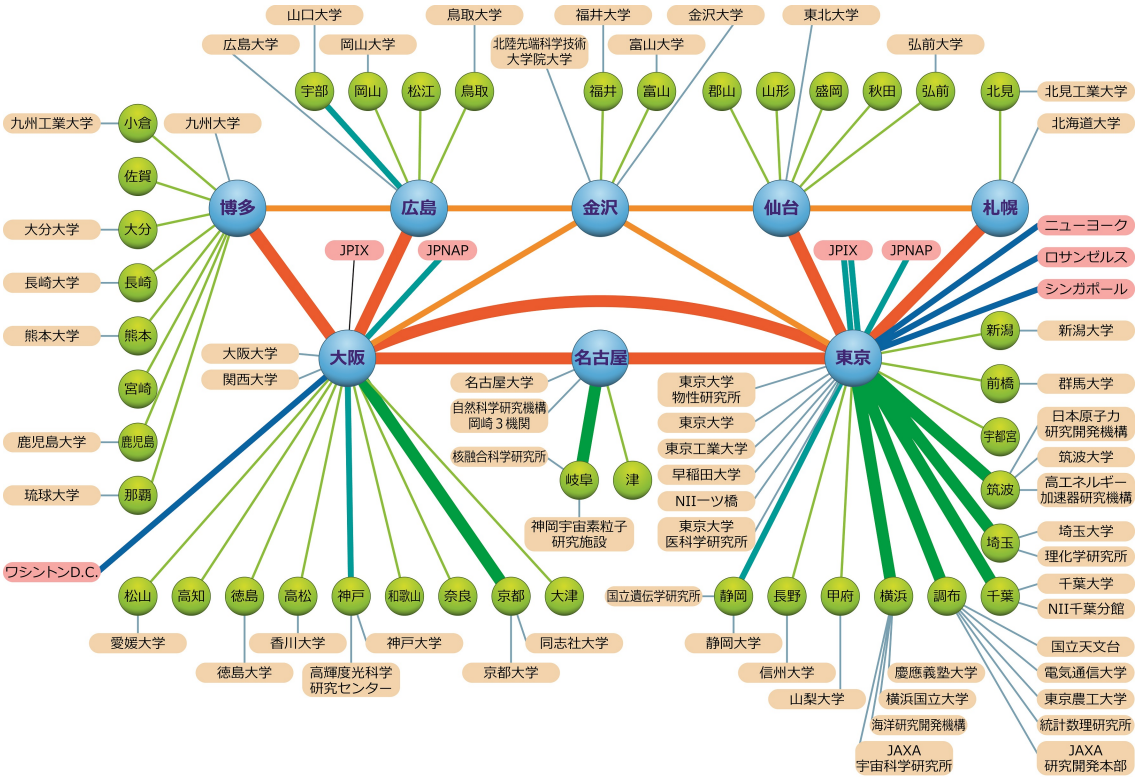

Fig. 3.2: SINET4 overview

3.3 Log time series creation

We construct log time series of the number of log appearances for each minute per event per day (i.e., data point). The processing flow is as follows (see also Figure3.3): After we classified raw log data into the template of logs per device, one event (for example, “ssh login at device 1") has 365 days long log entries. We first count the number of log appearances for each minute. Then, we split them for each days. We now have 465 data points of event “ssh login at device 1". In other words, each data point consists of one time series; It has 1,440 minute-length as X-axis and the number of log appearances as Y-axis (shown as x1 and x2 time series graphs in Figure 1).

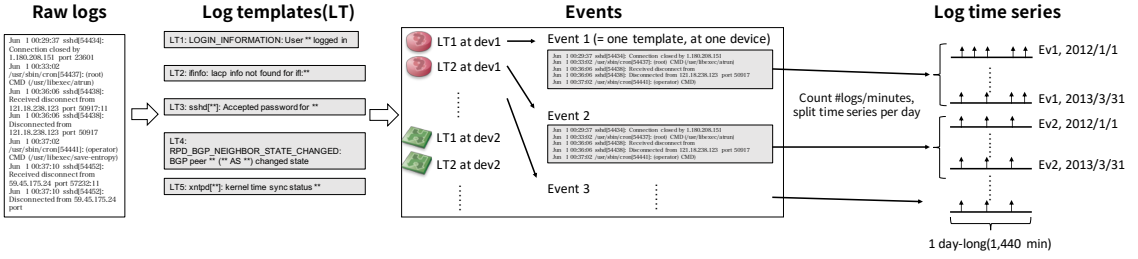

Fig. 3.3: Log time series creation flow

Chapter 4

Burstiness and causality analysis

In this chapter, we focus on burstiness of log time series and evaluate how useful the occurred burst is for network operation. The goal of this analysis is to evaluate usefulness of log burst for network management. In other words, a question focusing in this analysis is how observed bursts in log messages are useful for network operation. We conduct three analysis. First, we detect burstiness for each single log time series and evaluate the efficiency of our preprocessing. Second, we focus on cooccurred burst in two time series and analyze causal relationship between the pair based on causal analysis result. Lastly, we apply burst detection to log time series grouped by device as a multivariate log time series analysis.

4.1 Overview

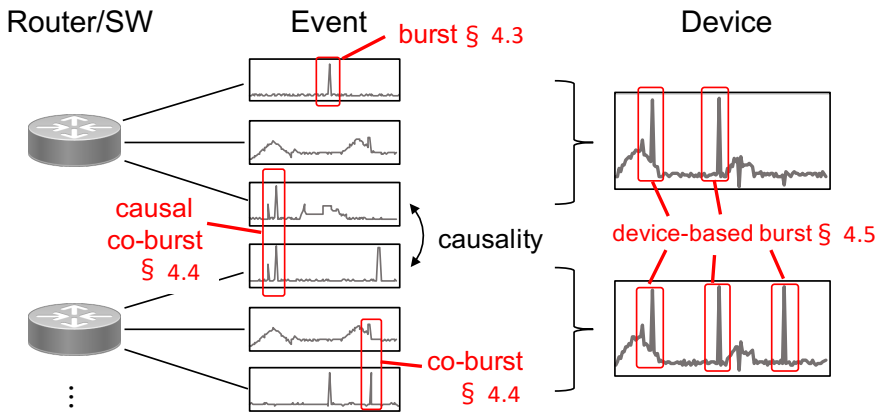

Fig. 4.1: Burstiness and causality analysis overview

Log burstiness is one of the most popular anomalous behavior in log appearances. In real network operation, network operators often pay attention to the burstiness of logs and try to use it as a clue for system anomaly detection. However, log bursts are occurred too frequent in real network system and they are not always related to the system anomaly. Thus, we first detect log

Table 4.1: SINET4 Dataset and analysis result

ID	logs	devices	templates	filtering	bursts
Data 1	34.7M	130	1789	no	400,518
Data 2	2.3M	130	1789	yes	32,704

bursts from real network logs and analyze usefulness of log burst for network management. In other words, a question focusing in this analysis is how observed bursts in log messages are useful for network operation.

To detect burstiness, we rely on Kleinberg's burst detection algorithm [37] and conduct three types of analysis depending on combination of the log time series: single, pair, and device-based burst detections (see also Fig. 4.1). We first apply the burst detection to the single log time series per event per day. After that, we investigate co-occurrence of bursts in a pair of such time series.We calculate burst co-occurrence ratio of pairs of bursts and compare with the causal inference results [35]. In device-based burst detection, we aggregate logs for each devices and apply burst detection. We focus on the bursts which can be detected only in this device level aggregation, but not in single event-based burst detection.

The findings of this analysis is as follows. First, in order to efficiently apply the Kleinberg's burst detection method to complicated and large scale network logs, we propose log time series preprocessing and it removes over 90% of trivial bursts. Next, with three types of analysis and comparison with causal inference results, we extract meaningful burst data for troubleshooting from all the detected ones. Third, in device-based burst detection, we find some causal bursts which are not found in existing causal inference results. To combine with these results, we extract meaningful log bursts from a lot of trivial ones for troubleshooting and also extract causal relationships between log bursts.

4.2 Preliminary

4.2.1 Removing trivial log messages

The majority of log messages are related to daily processes, such as cron, ssh authentication, and so on. These logs are not helpful for trouble shooting because they commonly appear and indicate daily process results. Furthermore, to process a large amount of such trivial data cause serious accuracy degradation and time consumption. In particular, the causal inference method is greatly affected by frequent logs. Also, the burst detection method requires more time for processing larger data. To remove such the frequent and unimportant logs, we applied a frequent data filtering method to original log messages. This process has two parts: (1) Remove periodic logs by a Fourier analysis, and (2) Remove highly frequent logs by linear fitting appeared in very short period. In processing (1), we applied the Fourier transform to the time series, removed the data whose peak of frequency spectrum exceeded a threshold, and applied the inverse Fourier transform to reconstruct the time series. In Processing (2), we applied the linear regression to a cumulative time series of the data and calculated the regression error. If the error is below a threshold, we removed the data. We combined these two approaches because the Fourier analysis may miss very short periodic data and the linear regression analysis may not remove long interval data. Finally, we obtained non-periodical log time series from the original log messages.

4.2.2 Burst detection

We detect log burstiness from log message time-series. The log burstiness is a local state of a large number of log appearances. Kleinberg's burst detection algorithm [37] is a well-known method to detect burstiness from stream data. We apply this algorithm to network log time series for burst detection.

Kleinberg's burst detection algorithm simulates burstiness states for each time series data using an infinite-state weighted automaton model. This method estimates the burstiness states by minimizing a state transfer cost function. It is a batch processing and detects relative burstiness in the input data. We briefly review the algorithm. It evaluates burstiness based on the interval

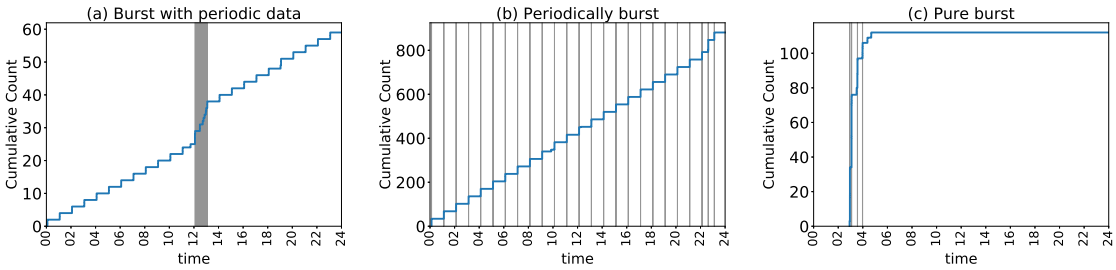

Fig. 4.2: Burstiness event examples

of the input data. The intensity of burstiness is given by burst level i. When the burst level is i in a state, the data arrival rate is distributed exponentially $f(x) = \alpha e^{-\alpha x}$. α is defined as $\alpha_i = \frac{n}{T} s^i$ where n is the number of input data and T is a time length of input data. s is a scaling parameter that indicates an arrival rate difference between neighboring states. Each automaton states have one burst level. State transition q is equal to maximizing the posteriori probability $P(q|x)$ where x is input data. Also maximizing $P(q|x)$ corresponds to minimizing a state transition cost function $c(q|x)$. $c(q|x)$ is represented as a sum of a threshold parameter γ and $p(x|q)$.

4.3 Single burst analysis

4.3.1 Methodology

We first analyze burstiness of single event time series with the burst detection algorithm. We divide the dataset to one-day long time series per event per device and detect burstiness in seconds. We implemented the detection algorithm by Python 3.6 and Pybursts, which is a python module of burst detection. We empirically set the scaling parameter $s = 2.0$ and the threshold parameter $\gamma = 1.0$. As the algorithm does not support a case of multiple events at the same time, we randomly shift these appearance time to 0.001 seconds (i.e., serialized). We use the minimum level of the burstiness for our analysis while the algorithm outputs several levels of the burstiness.

4.3.2 Result of burst detection

We apply the burst detection to each event of Dataset 1 and 2. As summarized in Table 5.1, we detected 400K bursts in the non-filtered data and 32K bursts in the filtered data. These correspond to about 800 and 70 bursts per day per event. In other words, only 7.5% of bursts are potentially meaningful. Figure 4.2 illustrates some examples of detected burstiness data. Each graph shows one-day cumulative time-series of an event. Gray rectangles indicate detected bursts.

We confirm mainly two types of burstiness; Burst with periodic data (Fig. 4.2(a)) and Pure burst (Fig. 4.2(c)). All of the bursts obtained in the dataset are not always useful for the network operation. For example, periodic burst due to daily processing are remained in Fig. 4.2(b). Indeed, this is a time series in Dataset 1 and this type of data is successfully removed in Dataset 2. These results demonstrate the importance of pre-processing, especially, removing the strictly periodic logs from the data. On the other hand, like Fig. 4.2(a) and Fig. 4.2(c), bursts with periodic data and pure bursts are important information for troubleshooting because changes of log appearances are a signal of system state changes.

4.4 Pair burst analysis

In this section, we intend to extract meaningful burst event pairs for troubleshooting from the results of the single burst detection using the co-occurrence of bursts and causality between events. First, we analyze the event co-occurrence related to burstiness using single burst detection results. Next, to analyze causality of burstiness, we compare the co-occurrence of bursts with the causal inference results. Finally, we classify causality of burstiness based on an event similarity using dynamic time warping distances [38].

4.4.1 Co-burst analysis

Methodology

We define Co-burst and a burst co-occurrence ratio to evaluate relevance between two bursts. Co-burst is a pair of two co-occurred bursts in two event time series. Co-burst event pair is a pair of two events which include at least one co-burst. We define the co-occurrence of two bursts if they start in the same 1-min bin. We calculate the co-occurrence ratio of events A and B as $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. $J(A,B)$ is the Jaccard similarity coefficient [*1]; The higher $J(A,B)$, the more frequent bursts occur at the same time between event A and B. A and B are events that are sets of time series data classified by each log templates, and $|A|$ and $|B|$ are the total number of bursts detected in 456 days. $|A \cap B|$ is the total number of co-bursts between event A and B in 456 days. $|A \cup B|$ is a disjunction of A and B.

Co-burst analysis result

We calculate the burst co-occurrence ratio using the single burst detection result (§4.3.2). The number of co-burst event pairs is 29,107 in Dataset 1 and 22,756 in Dataset 2. The distribution of the co-occurrence ratio is shown in Figs. 4.3 and 4.4. The vertical line shows the co-occurrence ratio in a log scale and the horizontal one the size of the union set of each event pair. The larger $|A \cup B|$, the more number of bursts is occurred in event A or B. Each dot shows a co-burst event pair.

In Dataset 1, 22% of dots are located in $|A \cup B| > 10,000$. These pairs are mostly periodic bursts, shown in Fig. 4.2 (b). As in §4.3.2, these bursts should be removed because they are the result of less useful daily processes. On the other hand, shown in Fig. 4.4, such larger points ($> 10,000$) disappeared in Dataset 2. This result confirms that the pre-processing in §4.2.1 works correctly and is effective to remove the trivial burst pairs.

4.4.2 Comparison with causality DB

Here, we compare the co-burst analysis results to the causality DB that includes 2,776 causal event pairs obtained in the prior work [35]. If the same event pair of co-burst events is recorded in the causality DB, we consider that the co-burst is not a coincident but a causal co-occurrence. We call such event pair as a causal co-burst. Table 4.2 shows the number of the causal co-bursts. The first row shows the total number of causal co-burst event pairs and the second one shows the number of events for $J > 0.1$. Red dots in Fig. 4.4 illustrate such causal co-burst events, while gray dots illustrate not causal but co-burst events.

Our results highlight two findings. First, we find a relationship between the causal co-burst events and the burst coefficient ratios. In Fig. 4.4, most red dots located at an upper part of the graphs, and over 89% of causal co-burst events have larger than 0.1 coefficient ratio from Table 4.2.

[*1] We also obtained similar results with the Simpson index.

Table 4.2: Co-burst and event causality

	Co-burst	**Causal Co-burst**
all	22,756	241 (1.06 %)
$J(A, B) \geq 0.1$	17,541	215 (1.23 %)

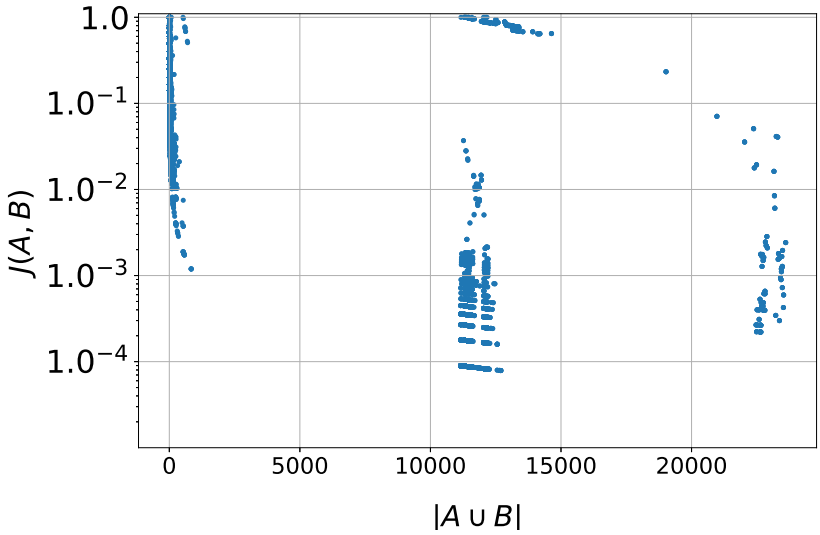

Fig. 4.3: Burst co-occurrence ratio (Dataset 1)

Focusing on horizontally distributed causal co-bursts (red points), we can see two clusters; a small burst disjunction cluster ($|A \cup B| < 500$) and large one ($|A \cup B| > 500$). Inspecting the detail of the large one, there are 1316 pairs and we confirm 487 pairs are derived from “show interface" command (i.e., ordinal operation). The others (829 pairs) are triggered with this command by chance. On the contrary, there are only 1 pair related to the command in the small cluster. The small cluster has 21,440 pairs and the number of the causal co-burst pairs is 167. There are 156 causal co-burst pairs in this cluster. Thus, we conclude that the causal co-burst event pairs have relatively high coefficient ratios and the frequency of their bursts is comparatively low. Therefore, meaningful causal bursts are located at the upper left cluster of red points in Fig. 4.4. Second, we confirm the effectiveness to combine the causal inference results with the burst detection. Combining the causal inference, we observe only 1% of them are remained. Therefore, 99% of co-burst event pairs are coincident and we have to focus on 1% of them for extracting meaningful bursts.

4.4.3 Time series similarity and co-burst classification

Here, we measure the time series similarity between causal co-burst event pairs and classify the type of co-burst towards practical system management.

Time series similarity and causality

Figures 4.5 and 4.6 visualize examples of temporal patterns of causal co-burst. Orange and blue plots indicate two types of events. Checking the details of the log events, we confirm these examples are helpful in network operation. In Fig. 4.5, login and authentication events in one device obviously have a causal relationship. Also in Fig. 4.6, an outage of a L2 switch invokes a bypass event, which means a change of network routes to avoid outage devices and keep network availability.

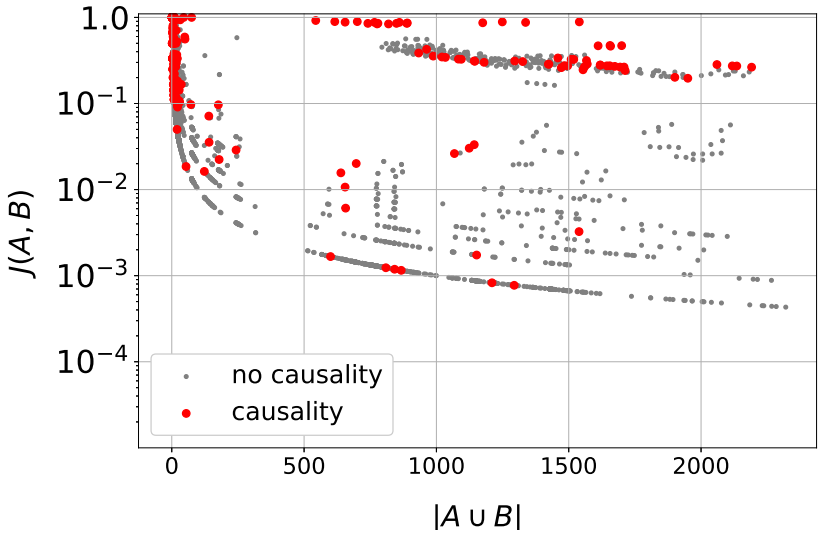

Fig. 4.4: Burst co-occurrence ratio (Dataset 2)

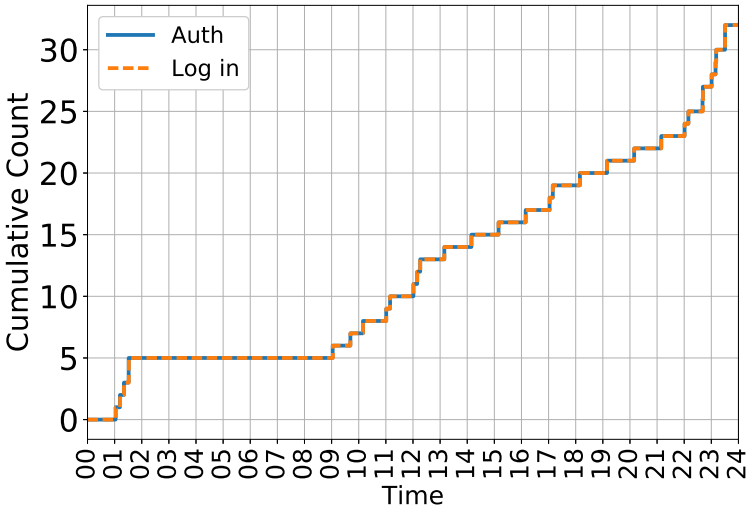

Fig. 4.5: Co-burst (from single device)

However, these two examples are different in whether they are triggered from one network activity or multiple ones. As shown in Fig. 4.5, log in activity always includes both log in and authentication event. On the other hand, in Fig. 4.6, a system down and enabling bypass events are not always appear at the same time. This is because system down and bypass event come from different network activity. Therefore, there are two types of event causality. (1) Causality from one network activity, and (2) Causality between different network activities. This difference also appears in time-series. Two events in the pair of type (2) exhibit almost the same pattern during a whole time-series, while the type (1) only shows the same pattern locally. Thus it is valuable to classify these two types of co-burst based on a similarity of whole time-series because we can distinguish whether causal co-burst is type (1) or type (2).

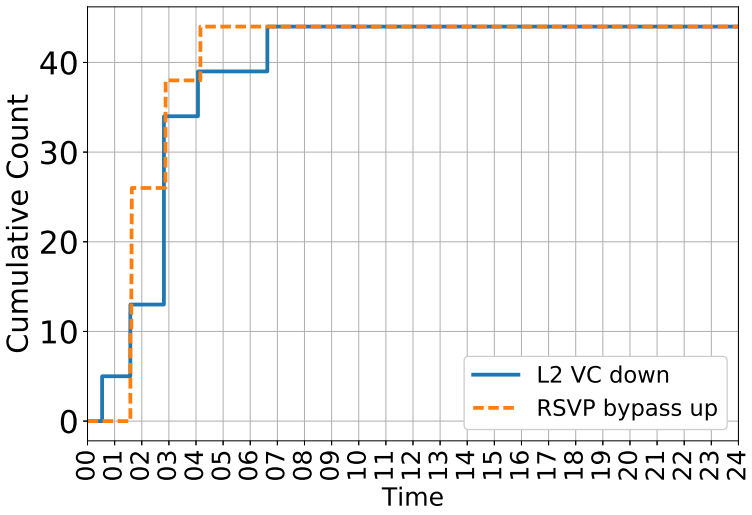

Fig. 4.6: Co-burst (from multiple devices)

Time series similarity analysis with Dynamic Time Warping distance

We introduce a similarity of temporal patterns of causal co-burst events by using Dynamic Time Warping (DTW). This similarity enables us to distinguish whether causal co-burst events come from different network activities or same one.

To calculate the similarity of two events, we should consider a case that two time series are mostly similar but partially different. Thus, the strict comparison of two time series may miss potentially similar events. DTW is a dynamic scheduling method to compare the similarity between two time-series. It can disregard the difference of periodicity and data size. First, DTW processes data points matching between two events which minimize distance of matching points. We choose the absolute distance of log occurrence time as an inter-points distance. Then, we aggregate all the inter-points distance as the DTW distance (similarity).

We apply this method to all the causal co-burst event pairs per day. We set the similarity threshold to 10,000, meaning that we permit up to 10,000 seconds-length (i.e., about 3 hours) errors in total per day. For each event pairs, if DTW distance is under threshold, we classify them as similar pairs and otherwise as dissimilar ones. The result is shown in Fig. 4.7. The horizontal line indicates manual classification of events and vertical one does the number of appeared days. In the each bar, blue part shows similar pairs and orange part shows dissimilar ones. There are 40 pairs of event classes and 13 pairs are classified dissimilar in more than half of appeared days (red labels). To focus on these 13 pairs, they include 9 pairs of two different classes. We consider these types of pairs as type (1) (see §4.4.3) and we successfully classify them with DTW distance. Therefore, we can distinguish whether a causal co-burst comes from type (1) or type (2) based on time series similarity. This metrics is useful to evaluate the complexity of causality.

4.5 Device-based burst detection

Previous sections focused on the burstiness appeared in time series per event per device. Here, we analyze time series aggregating all events per device.

Fig. 4.7: DTW distance distribution. Red labels show dissimilar pairs classified by DTW distance.

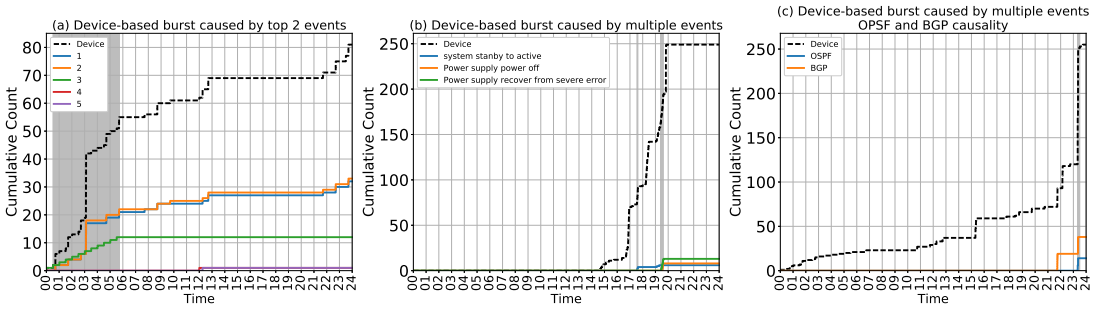

Fig. 4.8: Device-based burst examples

4.5.1 Device-based log burstiness

We aggregate events per device and apply the burst detection to them; The detection method and its parameter setting are the same with the single burst detection. There are 130 devices and 1,789 templates in Dataset 2. We aggregate events across all the templates in each devices (i.e., 130 devices $\times$ 456 days). We detect 21,670 bursts and refer to them as device-based bursts.

We find two types of device-based burst by analyzing major templates contributing to device-based bursts. One is the burst that is also detected in each events (§4.3.2) and the other is only detected in this device-based analysis. Figure 4.8 (a), (b) and (c) shows an example of the device-based burst. The black line shows device-based time-series. Other lines show some event time series in the device.

Commonly, one or two events dominate a majority of the device-based logs in Fig. 4.8(a). Thus, some device-based bursts appear due to a few dominant events and these bursts have been similarly detected in the single burst analysis (§4.3.2). We can see that a burst starts in two major events (events 1 and 2) as well as the device-based burst at 3:00. On the contrary, a device-based burst also appears when many types of small events occur at the same time. In Fig. 4.8 (b), we can see a device-based burst at 20:00 without such majorities. We further focus on the latter type of bursts (such as shown in Fig. 4.8 (b) and (c)) because the previous event-based analysis missed finding them.

4.5.2 Device-based burst detection

We extract the bursts that do not appear in the event-based analysis from the device-based bursts. Here, we focus on such 3,735 bursts extracted from 21,670 bursts.

To investigate events contributing device-based bursts, we dig into what happened at that time. Figure 4.8 (b) shows an example of a device-based burst. Observing many events related to system standby and power supply occurred at the same time, we understand a system reboot happened on this device due to some reason.

In another example (Fig. 4.8 (c)), a device-based burst appears from 22:00 to 23:00 as well as many OSPF and BGP events. This suggests some routing errors. Note that this burst cannot be detected by the event-based analysis, thus, the device-level aggregation is useful as well as the event-level analysis.

4.5.3 Comparison with causality DB

To analyze the causality between events in device-based bursts, we compare them with the causality DB. Because we only know causality between two events, not multiple events from the

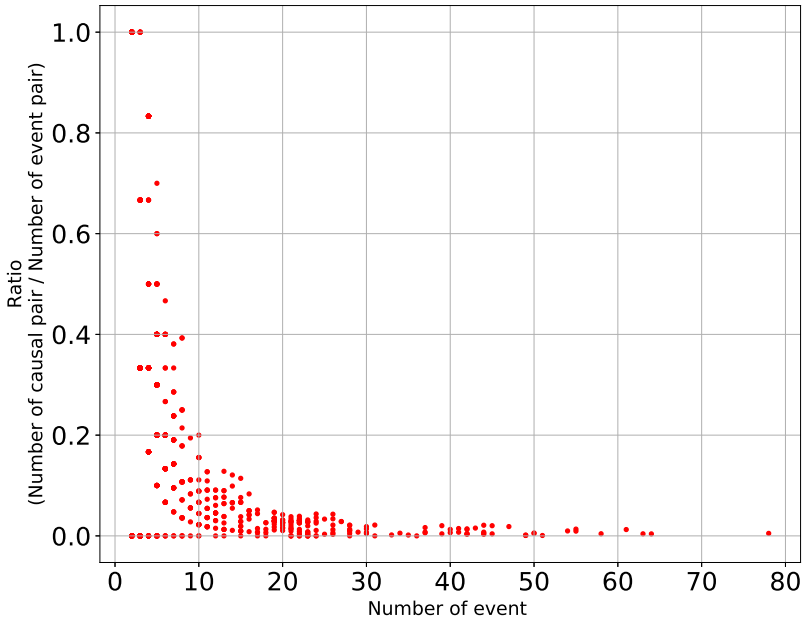

Fig. 4.9: Device-based burst distribution

causality DB, we inspect whether the event pairs have causality. Figure 4.9 shows the ratio of the number of causality event pairs to all the pairs of events contributing device-based bursts. The vertical axis indicates the ratio: $\frac{\text{CausalEventPairs}}{{}_{Events}\text{C}_2}$. The horizontal line shows the number of events and each point corresponds to device-based bursts. We confirm that most events composing device-based bursts do not have causality. However, we find some cases in which events in the device-based burst have causality considering from log contents, not from the causality DB. Figure 4.8 (c) is such an example. A device-based burst is appeared in around 22:00-24:00 and both OSPF and BGP events contribute to the device-based burst. The two events obviously have causality in terms of network operation. However between BGP and OSPF events, time-series causality is not recorded in the causality DB. Therefore device-based burst analysis can detect causality which cannot be detected by time-series causal inference method and device-based burst analysis can improve the causality analysis.

4.6 Summary

We focused on burstiness and causality appeared in network log messages. We conduct three types of analysis: single, pair, and device-based burst detection. To combine burst detection results, causal inference results and other metrics such as burst co-occurrence ratio and dynamic time warping distance, we reduce a large amount of trivial bursts, and extract meaningful information from log messages. From single burst analysis, only 7.5% of bursts are potentially meaningful and we can remove rest of 92.5% of meaningless bursts by our time series preprocessing. Combined with causality DB, we find that only 1% of co-occurred bursts have true causality. Our similarity analysis separates whether the causality of co-burst comes from one network activity or multiple ones. In the device-based burst detection, we find 3,735 bursts that are only found by this multivariate analysis. Also we find some causal bursts which are not found in the existing causal inference results.

These analysis highlight following findings: preprocessing based on heuristic is highly efficient for removing trivial anomalies for network operation. Even applying preprocessing, there are too many cooccurred bursts and about 99% of them are coincident. We have to focus on the rest of 1% of cooccurred bursts which have causality and the cooccurred ratio is potentially useful metrics for judging usefulness for network operation.

Chapter 5

Latent variable based anomaly detection

In this chapter, we propose latent variable based anomaly detection in log data and evaluate our proposed method. In chapter §4, we combine burst detection which is one of the most typical detection algorithms and other metrics.

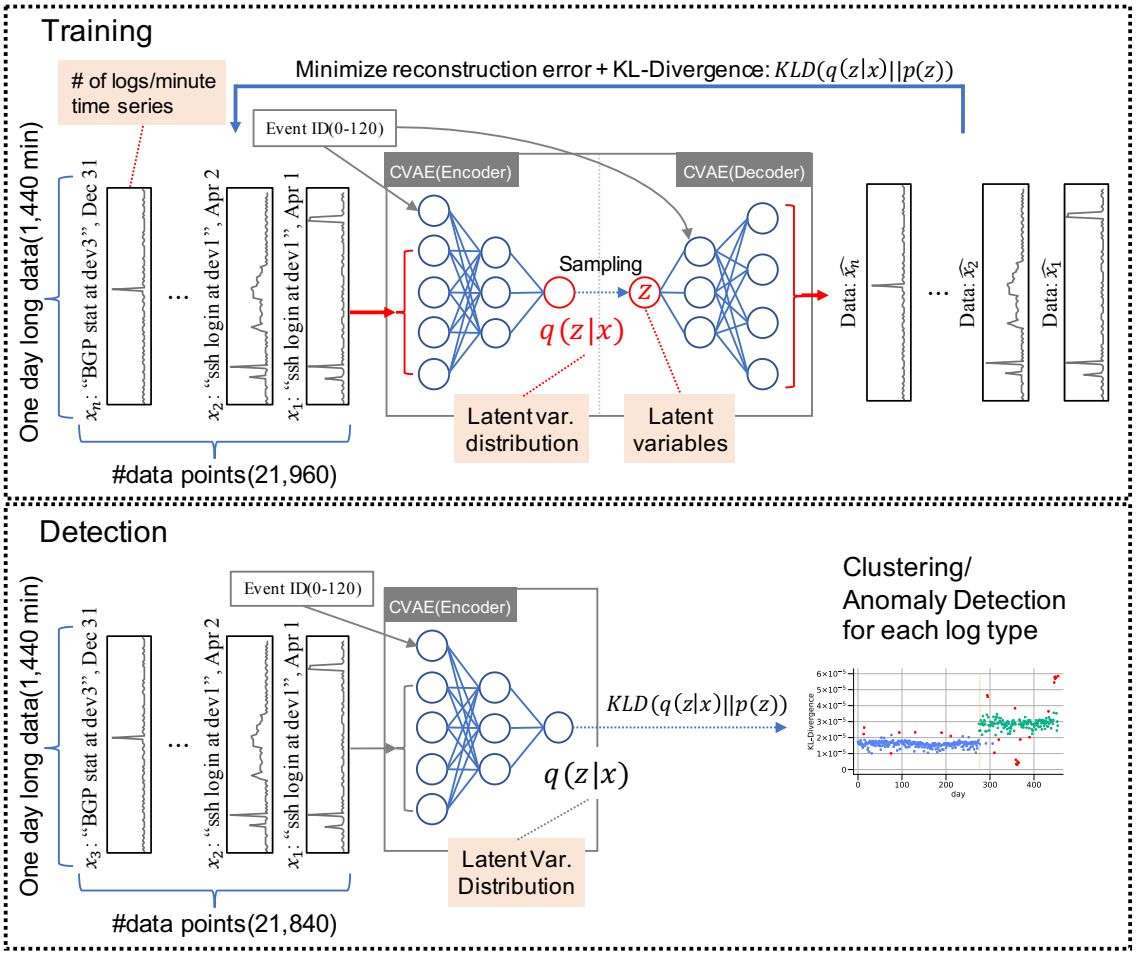

Fig. 5.1: Methodology overview

5.1 Overview

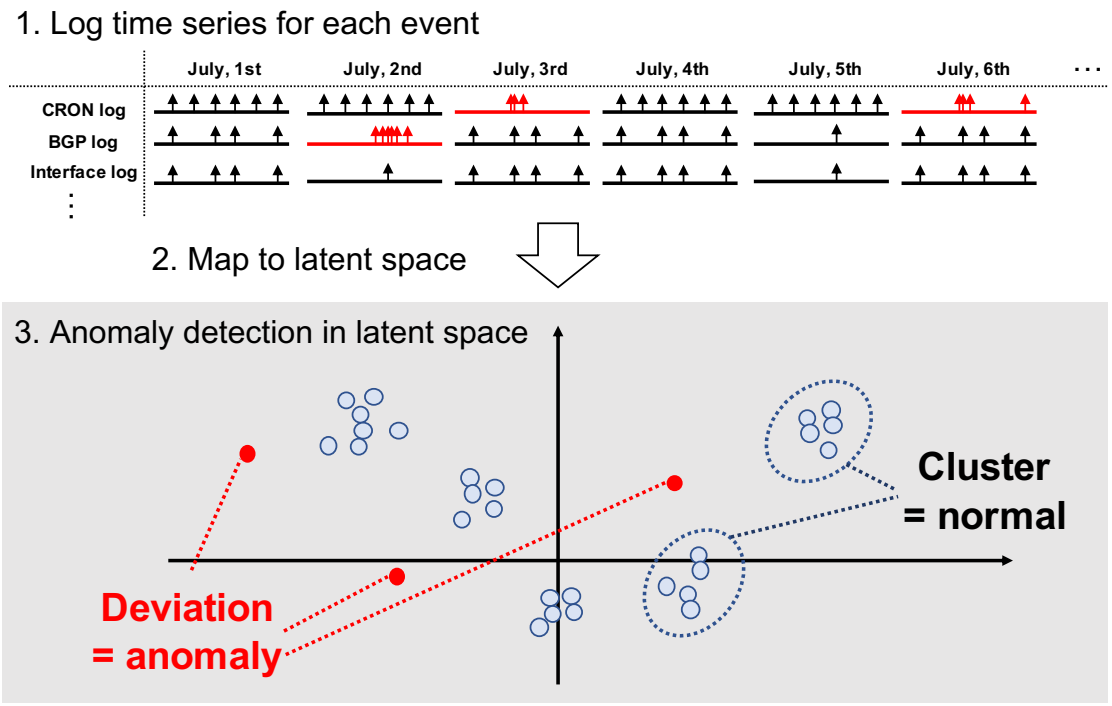

Fig. 5.2: Key idea of latent variable based log anomaly detection

Next, for more general log analysis, we propose a latent variable based anomaly detection method, which does not rely on any statistical preprocessing or predefined anomaly. We define an anomaly as time series behavior deviating from the normal or usual state in log time series. To detect these anomalies, we propose a robust statistical method that handles complicated system log time series, without any specific (case-by-case) preprocessing (e.g., filtering, smoothing, auto-regression) for log anomaly detection. The key idea of our approach is to embed a large amount of diverse data into hidden states by using latent variables and detect anomalies in latent space (as shown in Fig 5.2). A latent variable analysis is also known as topic modeling or Latent Dirichlet Allocation (LDA) often used in natural language processing. A topic model discovers topics as latent variables from a collection of documents. We borrow the idea of this approach: We first translate raw log messages into log time series for each type of logs (step (1) in Fig 5.2). Next, log time series are mapped into latent variables per day per type of logs (step (2) in Fig 5.2). In the latent space, values of latent variables represent the trend of the corresponding time series. After that, we apply a clustering method to latent variables and detect deviations from detected clusters in the latent space (step (3) in Fig 5.2). We consider clusters in the latent space are groups of normal trend time series in the real log time series. On the other hand, deviations from clusters in the latent space are interpreted as anomalies because the deviation in the latent space means deviated time series trend in the real log time series. Thus, with latent variables, time series can be represented by their trend without any domain knowledge or preprocessings to real log time series.

Our proposed method consists of two phases (see also Fig 5.1): (1) The training phase embeds log time series characteristics into latent variables by using Conditional Variational Autoencoder (CVAE). To mitigate the difficulty in handling various time series from many devices together, we

provide CVAE with type of log messages as a conditional label. (2) The detection phase highlights anomalies deviating from the distribution of the latent variables with clustering algorithms.

Using syslog data collected in a nation-wide academic network in Japan (SINET4), we confirm through evaluation that CVAE has higher discriminative power for analyzing complex time series than Kleinberg's univariate burst detection [37] and a traditional multivariate analysis (Principal Component Analysis; PCA). With a case study, we demonstrate that our method is useful in troubleshooting network system faults.

The contribution of this work is twofold. We first propose our latent variable-based method for highly diverse log time series data without any data-specific preprocessing. Next, we discuss the effectiveness of the method when we applied it to real network syslog data.

5.2 Training phase

We briefly explain CVAE, a variation of the Variational Autoencoder (VAE). The VAE [39] is a stochastic variational inference method based on the variational Bayesian approach. Since this is an unsupervised method, annotations for logs are not necessary.

Let us consider one-day long log time series $\mathbf{x}$ generated by a random process based on invisible continuous random variable $\mathbf{z}$. This $\mathbf{z}$ is also subject to a prior distribution $p_\theta(\mathbf{z})$. Thus, $\mathbf{x}$ is subject to a conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$. The goal is to obtain latent variables $\mathbf{z}$ from input $\mathbf{x}$. In order to get the latent variables, we estimate the distribution $p_\theta(\mathbf{z}|\mathbf{x})$ with Bayes' theorem and approximate distribution $q_\phi(\mathbf{z}|\mathbf{x})$. Now, the objective is to maximize the marginal likelihood (MLH) $\log p_\theta(\mathbf{x}) = D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) + \mathcal{L}(\theta, \phi, \mathbf{x})$. This equation is rearranged as follows (details are given in [39][40]).

$$\log p_\theta(\mathbf{x}) \geq \mathcal{L}(\theta, \phi, \mathbf{x}), \tag{5.1}$$

$$\begin{aligned}\mathcal{L}(\theta, \phi, \mathbf{x}) = &-D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) \\ &+\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})].\end{aligned} \tag{5.2}$$

We train the model that generates $q_\phi(\mathbf{z}|\mathbf{x})$ from input $\mathbf{x}$ by optimizing the lower bound $\mathcal{L}(\theta, \phi, \mathbf{x})$. The training process is as follows. First, we consider the prior distribution as a Gaussian $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ and let $q_\phi(\mathbf{z}|\mathbf{x})$ be a multivariate Gaussian. Next, we estimate the parameters (ϕ) of the posterior distribution ($q_\phi(\mathbf{z}|\mathbf{x})$) with a fully connected neural network (encoder). Then, we acquire the distribution of the latent variable $q_\phi(\mathbf{z}|\mathbf{x})$, and through the sampling process from it, we obtain the latent variable $\mathbf{z}$. Finally the decoder neural network reconstructs $\hat{\mathbf{x}}$ from the latent variable $\mathbf{z}$, and we feedback the loss values in Eq. (2). Reviewing Eq. (2), we can consider the first term (Kullback-Leibler divergence, KLD) as the distance between estimated distributions $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{z})$, and the second term (Marginal Likelihood, MLH) as the reconstruction error between $\mathbf{x}$ and $\hat{\mathbf{x}}$. Through these processes, we can compute Eq. (2) and proceed with the training of the encoder and decoder neural networks.

The CVAE [40] is a variation of the VAE. CVAE uses label information when generating the latent variable $\mathbf{z}$ and reconstructing $\hat{\mathbf{x}}$. We define the conditional label as the event that is an index of a unique combination of devices and log templates.

5.3 Detection phase

After applying CVAE to log time series data, we obtain the latent variables for each data. Now, as the latent variables well-describe time series trends, the goal of this phase is to build upon the time series description by latent variables to find the deviation from "normal" states.

Some latent analysis methods define the anomaly level based on reconstruction errors [28][20]. In our case, we can use MLH for reconstruction errors. However, there are two problems with using

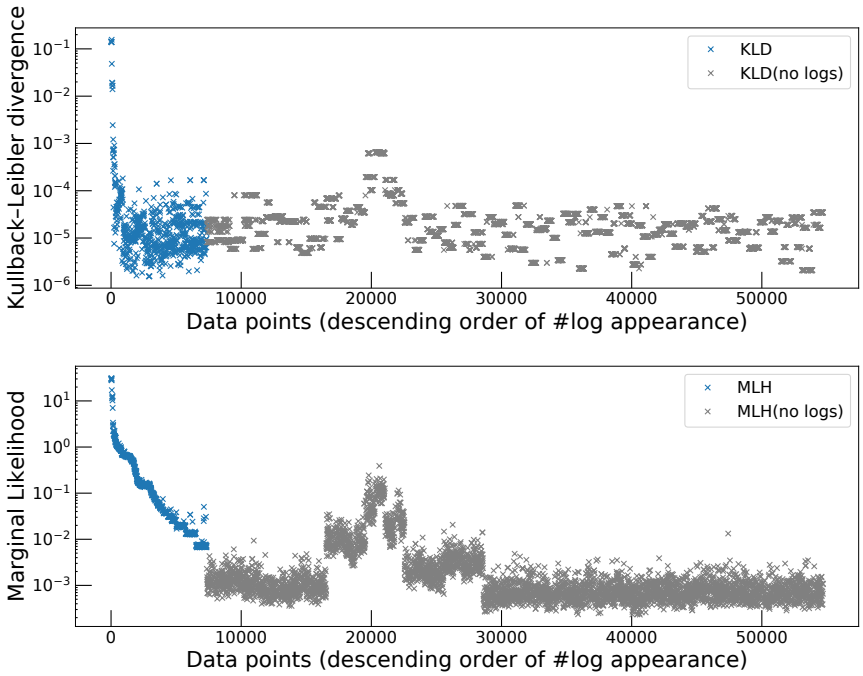

Fig. 5.3: Kullback Leibler divergence and Marginal Likelihood

MLH. (1) The reconstruction process is stochastic and MLH values follow a Gaussian distribution. (2) MLH is biased by the intensity of the original data.

To solve these problems, we rely on the distance between latent variable distribution ($q_\phi(\mathbf{z}|\mathbf{x})$) and the assumed prior distribution ($p_\theta(\mathbf{z})$) by computing $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}))$ in Eq. (2). After training, the learned model generates the distribution of the latent variable ($q_\phi(\mathbf{z}|\mathbf{x})$) from input data $\mathbf{x}$ as the output in a hidden layer (shown at the bottom of Figure 1.1). We use this distribution $q_\theta(\mathbf{z}|\mathbf{x})$ for detection, not the sampled value $\mathbf{z}$. Then, we deterministically compute the KLD ($D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}))$). The KLD represents the trend in the input time series $\mathbf{x}$. Thus, input data with similar KLD value belong to the same trend. Therefore, we can obtain reference behavior by clustering KLD values. Note that the KLD value is not an anomaly level but a state of input time series. Thus, we consider the data whose KLD value deviates from others as an anomaly regardless of the magnitude of numeric value of the KLD.

Fig 5.3 shows an example of the KLD and MLH values of the dataset. The horizontal axis shows data points in descending order of the number of log appearances, and the vertical axis shows the KLD (top) and MLH (bottom) values. In particular, the gray marks indicate days without any log appearances (i.e., 1,440-long zero vector), and the blue ones show more than equal to one log appearances on that day. Even if we input no log appearances data (i.e., gray marks in the Figure) to CVAE, KLD values are different for each event thanks to the weight of the conditional label (i.e., event ID). It enables such data points to belong to appropriate clusters depending on the behavior of the event. As shown in the figure, the MLH values are more spread and biased by the number of log appearances. In addition, they increase along with the number of log appearances. On the other hand, the KLD values are not biased by the number of log appearances and are more stable than MLH values. Therefore, we use the KLD as an indicator of time series trend.

Fig 5.4 is a cumulative distribution of normalized KLD values (blue curve) and normalized MLH values (orange curve). Red lines show 99% points in KLD and MLH. As shown in this figure, KLD requires the range from 10^{-7} to 10^{-2} to represent 99% of data points. However, MLH needs 10 times wider range than KLD (from 10^{-7} to 10^{-1}). Thus, KLD is more suitable for the

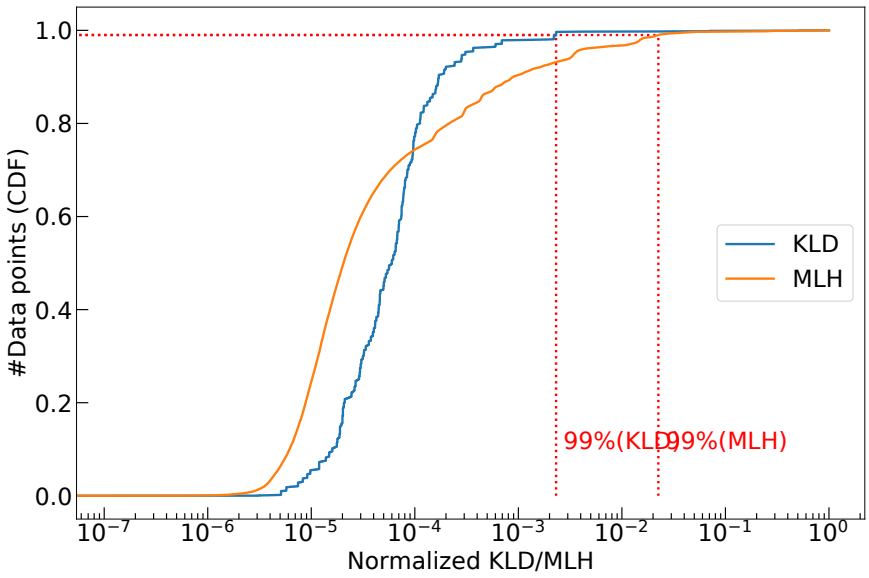

Fig. 5.4: Kullback Leibler divergence and Marginal Likelihood (CDF)

indicator of time series trend because dense data is easier to handle than sparse data.

Once latent variable distribution is obtained and the KLD is computed one can choose any clustering method for anomaly detection. In this study, we used a well-known density-based clustering algorithm, Density-Based Algorithm for Discovering Clusters (DBSCAN) [41]. By applying DBSCAN to the KLD time series for each event, we detect clusters of KLD (i.e., normal state of that type of log) and anomalies that do not belong to any clusters.

Table 5.1: SINET4 Dataset for anomaly detection

	Training	Testing
logs	174,623	284,317
devices	24	24
templates	33	33
events	120	120
data points	21,960	21,840
start	'12/04/01	'12/10/01
end	'12/09/30	'13/03/31

5.4 Evaluation

In this section, we discuss the evaluation of our method using 365 day-long system log data obtained from SINET4. First, we compare the performance of our method with PCA and burst detection technique using 6 month-long training dataset and 6 month-long testing dataset. Next, we evaluate dependency of training data set size on the accuracy of anomaly detection. Finally, we present a case study that demonstrates the effectiveness of our method for network troubleshooting.

5.4.1 Dataset and comparison algorithms

Dataset

We use a set of network logs collected at SINET [36], a Japanese research and education network. This network connects over 800 academic organizations in Japan and consists of eight core routers, 50 edge routers, and 100 layer-2 switches. We selected a part of 365 days data from 2012/4/1 to 2013/3/31 gathered from commodity L2 switches and L3 routers.

As log messages are string data, statistical techniques cannot be applied directly. Thus, we generate log templates from raw log messages with the supervised learning approach proposed by Kobayashi et al. [8]. Log templates are log messages without variables (e.g., IP address), as shown in Fig 3.1. We then classify logs into log templates and extract time series data from their time stamps for each template per device. We thus generate 1,789 unique log templates from the whole dataset. We define an *event* as 365 of 1-day log time series, which have the same log template in one device. For our analysis, we manually selected 120 events and applied CVAE to them (shown in Table 5.1) to make sure that the dataset has several anomalies and the diversity of the time series. We confirm that these selected events include a large variety of time series trends such as periodicity, burstiness, and sparseness as well as diverse categories of log templates shown in Fig 5.6.

We construct log time series of the number of log appearances for each minute per event per day (i.e., data point). The processing flow is as follows: After we classified raw log data into the type of logs per device, one event (for example, "*ssh login at device 1*") has 365 days long log entries. We first count the number of log appearances for each minute. Then, we split them for each days. We now have 365 data points of event "*ssh login at device 1*". In other words, each data point consists of one time series; It has 1,440 minute-length as X-axis and the number of log appearances as Y-axis (shown as x_1 and x_2 time series graphs in Figure 1). We also use conditional labels built on event labels concatenating log time series. To input event labels into CVAE, we first assign unique IDs to all the events and then encode them to one-hot vectors. If the "*ssh login at device 1*" event has ID i, the conditional label is a 120 long vector and values are zero except index i. The value of index i is one. Then, the conditional label has 120 dimensions. After repeating the same process to all the 120 events, we finally obtain $120 \times 365 = 43,800$ data points and conditional labels.

In the end, we explain training and testing data for our analysis. The first three months of the data include a lot of unusual behavior because this period was the migration and beginning of network operation in SINET4. Thus, we exclude the data in the first three months (10,920 data points in total) and use the rest of 43,800 data points. Then, we split this dataset into training dataset and testing dataset (see also Table 5.1). The training dataset is the first half of dataset (21,960 data points, that is from 2012/4/1 to 2012/9/30) and the testing dataset is the second one (21,840 data points, that is from 2012/10/1 to 2013/03/31).

Baseline algorithms

To understand the effectiveness of CVAE for log anomaly detection, we compare it to two traditional baseline algorithms: Kleinberg's burst detection [37] and PCA. The three algorithms are summarized in Table 5.2.

Table 5.2: Comparison of methods

name	multivariate	determin.	linear trans.
burst detection	no	yes	-
PCA	yes	yes	yes
CVAE	yes	no	no

Burst detection

Otomo et al. [42] conducted log analysis focusing on the burstiness and causality of log time series. They first removed trivial logs (e.g., periodic and very frequent logs) to prevent detecting trivial bursts then detected log bursts with Kleinberg's burst detection [37]. Applying the same algorithm to our dataset, we confirm 188 log bursts out of all 21,840 data points. In the following comparison, we use these bursts as a part of reference of log anomalies.

PCA

To detect traffic anomalies, Principal Component Analysis (PCA) is a well-studied algorithm [28][20]. It translates a set of input traffic time series data into main and residual subspaces with a linear transformation. This algorithm is similar to CVAE mapping raw data to the latent space, though CVAE is a non-linear transformation. We implement a PCA-based anomaly detector. Applying PCA to a set of log time series, we obtain principal components of the input dataset and separate them into main and residual principal components with a threshold based on their variance coverage. We then reconstruct each time series with the main components and compute the reconstruction errors with the original data. Finally, we apply a clustering method (DBSCAN) to these reconstruction errors and obtain anomalies.

5.4.2 Parameter tuning

We first briefly describe the parameter tunings of CVAE. Encoder and decoder networks in CVAE each have four hidden layers. The input layer has 1,560 dimensions (1440 dimension of time series and 120 dimension of label data). The encoding neural network has four layers, and each layer has 512, 256, 128 and 64 units in order from the input layer. We set the latent variables dimensions to 10. The decoding neural network has the opposite structure of the encoder, but the output size is 1,440 so that the output is a reconstructed time series. To avoid over fitting, we dropout 30% of units for each layer during training. During training, we empirically set the number of epochs to 50 based on test trial results. We confirm that loss values converge after training. Next, we compute KLD for each time series using the learned encoder network and apply DBSCAN to KLD for each event. We tune DBSCAN parameters for each event so that a size of cluster is longer than a week. As the training process is stochastic, the results are not the same even if the loss values are converged. To mitigate this uncertainty, we train the model ten times and adopt anomalies detected in majority votes.

We use a commodity computer (Xeon CPU and GTX-1080 GPU) for processing. One training takes 489.5s on average of 10 trials. The detection phase requires 45.1s on average to calculate KLD and conduct DBSCAN.

Table 5.3: Summary of results

	PCA	CVAE
Data	21,840	21,840
Anomalies	1,044	1,215
Precision	88.4%	91.5%
Recall	59.4%	74.0%
#Bursts	169 (89.9%)	170 (90.0%)

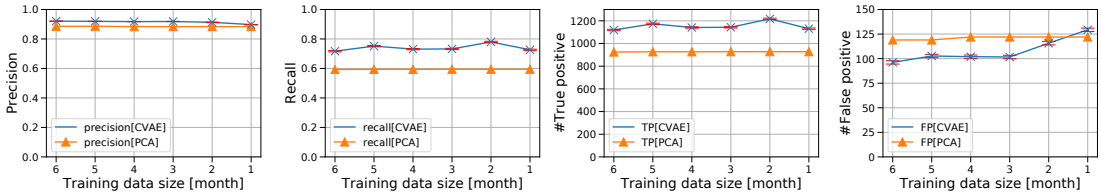

Fig. 5.5: Performance dependency on different training data size: Precision, Recall, #True positive and #False positive, from left to right, respectively.

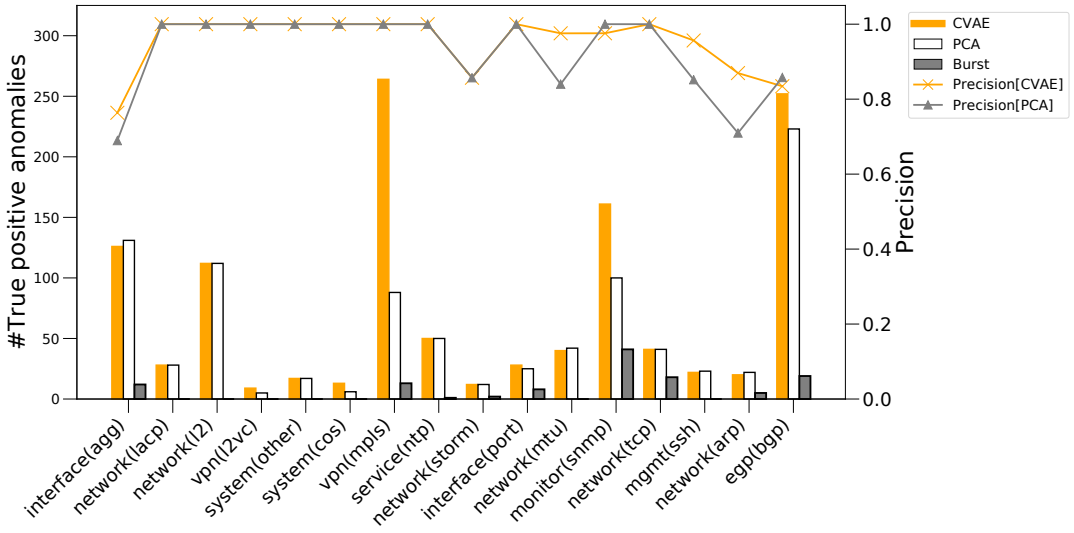

Fig. 5.6: Number of detected anomalies and their precision per log category

5.5 Result overview

For better understanding detected anomalies in network management, we manually inspected all the test data points and annotated them as anomaly or normal based on the SINET4 trouble ticket data and our domain knowledge for network management; For example, "*ssh login*" logs have periodic appearance in our data because they are generated by a CRON process including hourly remote ssh access. In this case, we consider periodic appearance as "normal" and lack of periodicity as "anomaly".

We first show that CVAE outperforms PCA (Table 5.3). Note that CVAE's results are averages of 6 trials because our proposed method is stochastic. CVAE detects more number of anomalies with higher accuracy than PCA; 1,215 anomalies with CVAE and 1,044 with PCA. CVAE's recall is about 14.5% higher than PCA's recall and CVAE's precision is about 3% higher than PCA's one. The number of overlap anomaly was 898.

We also confirm that CVAE and PCA detected $\approx$ 90% of the bursts detected by Kleinberg's burst detection algorithm as shown in the last row in Table 5.3. By inspecting missing bursts, we find two parameter tuning issues: (1) When similar burst patterns last for a few days, DBSCAN extracts them as a cluster, not outliers. The algorithm thus does not detect these bursts as anomalies. (2) Intensive bursts cause CVAE and PCA to detect a relatively small burst as a normal.

Fig 5.6 is a histogram of detected anomalies per log category. Orange bars indicate true positive anomalies detected with CVAE, white ones with PCA, and gray ones with burst detection. Orange line shows CVAE precision and white one shows PCA precision. In most cases, we find that CVAE

correctly detects anomalies such as burstiness and lack of periodicity. Focusing on the case in which PCA detects more true positive anomalies than CVAE (e.g., interface(agg)), we find CVAE has a higher precision than PCA thanks to the CVAE's robustness against noise (see § 5.7 5.7). On the other hand, when CVAE detects more true positive anomalies than PCA (e.g., vpn(mpls), monitor(snmp), egp(bgp)), we find that they still keep high precision thanks to its robustness against outliers (see § 5.7 5.7). Therefore, our proposed method is better balanced between the number of detected anomalies and precision than the traditional PCA-based method.

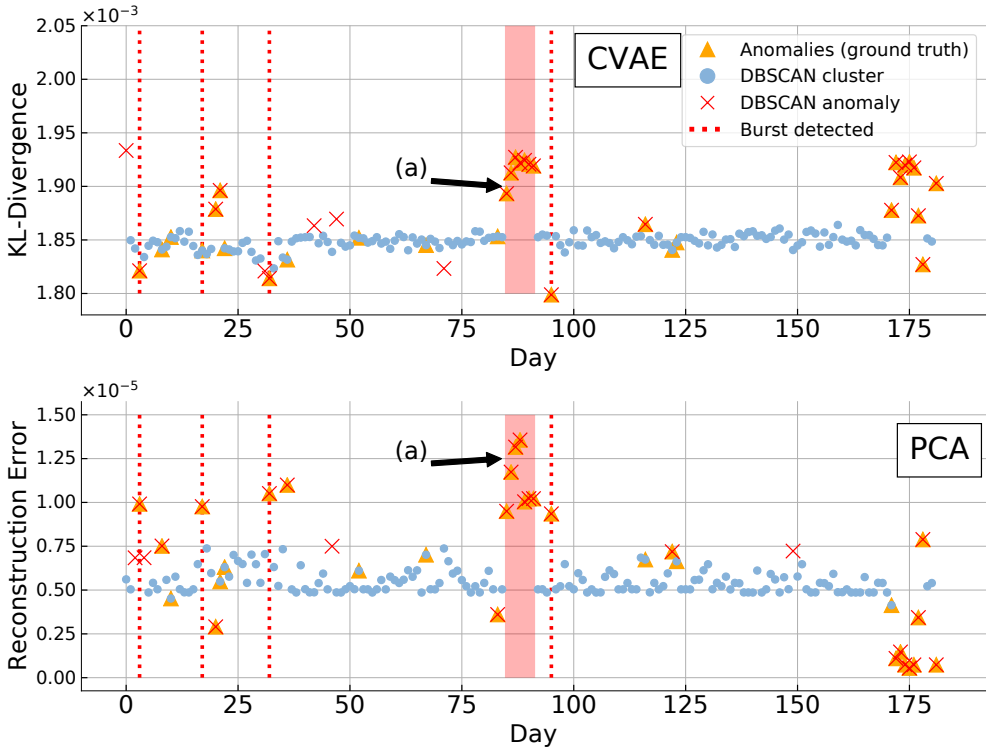

Fig. 5.7: Comparison between KLD (CVAE), reconstruction errors (PCA) and burst detection results

5.6 Dependency of training data size

We also evaluate the performance of our proposed method using different training data size: 1 - 6 month-long. As shown in Table 5.1, we have 6 month-long training dataset and 6 month-long testing dataset, which are continuous in time series. We use different size of training dataset in the order of newest date: 1 month-long dataset is from '12/9/1 to '12/9/30, 2 month-long dataset is from '12/8/1 to '12/9/30, and so forth. Table 5.5 shows detailed results for each size of training data. Blue lines are CVAE's results and orange ones are PCA's results. CVAE's results are averages of 3 trials in this figure. As shown in Table 5.5, CVAE's precision and recall outperform PCA's for each training data size. Focusing on the number of false positives (shown on the right in the figure), CVAE's one increases in less than 3 month-long training data. In the worst case, the number of false positives is larger than PCA's one in 1 month-long training data. Thus, we conclude that CVAE requires 3 month or more training data for obtaining appropriate results.

5.7 Detailed comparison

■Anomalies hidden in periodicity: Fig 5.7 shows the detailed results of periodic remote access logs for 182 days. The top graph shows KLD computed with CVAE and the bottom graph illustrates reconstruction errors obtained with PCA. Red dotted lines indicate anomalies detected by burst detection. These logs appear once per hour due to an automated remote monitoring script. We confirm significant outliers in the figure due to missing periodic logs (label (a)). As non-periodic anomalies represent failures of automatic remote access login, we intend to detect these non-periodic anomalies in addition to burst anomalies. As expected, the burst detection finds only burst anomalies. As shown in this figure, CVAE and PCA correctly detect these two type of anomalous data from others. Thus, CVAE and PCA correctly learn event-wise features (in this case, periodic appearance) thanks to their discriminative power.

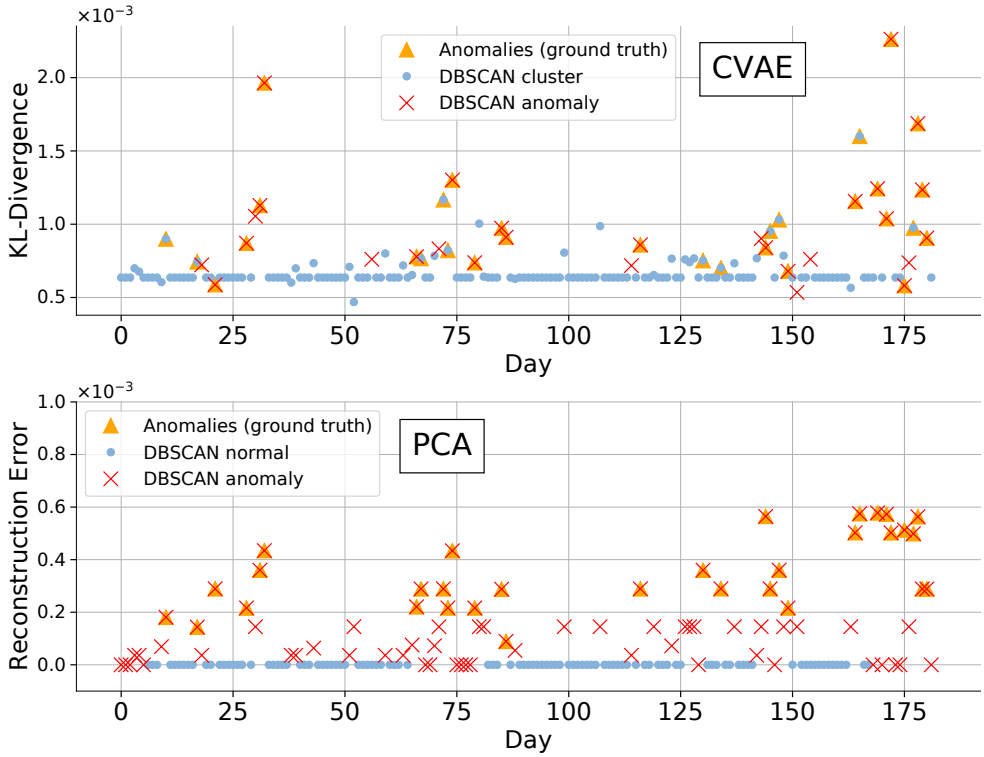

Fig. 5.8: Robustness of CVAE against noisy logs

■Robustness against noisy logs: If there are noisy logs without outliers, PCA yields many false anomalies due to its noise-sensitivity. However, as CVAE is stochastic and merges the multiple results, we can mitigate the effect of noises. Fig 5.8 (interface aggregate warning event) shows an example of this case. The top graph shows KLD in CVAE, and the bottom graph shows the reconstruction errors in PCA. The solid blue circles are normal points and the crosses are anomalies based on DBSCAN clustering. The orange triangles are true anomalies. This event has discreteness in the number of log appearance because its logs always appear with multiple lines (i.e., #network interfaces in its device). As shown in the figure, PCA is affected by this trend and also shows discreteness in reconstruction error. This trend makes anomaly detection or clustering difficult because errors spread with constant interval. With CVAE, however, the KLD values are relatively spread compared with reconstruction errors, and DBSCAN detects other anomalies.

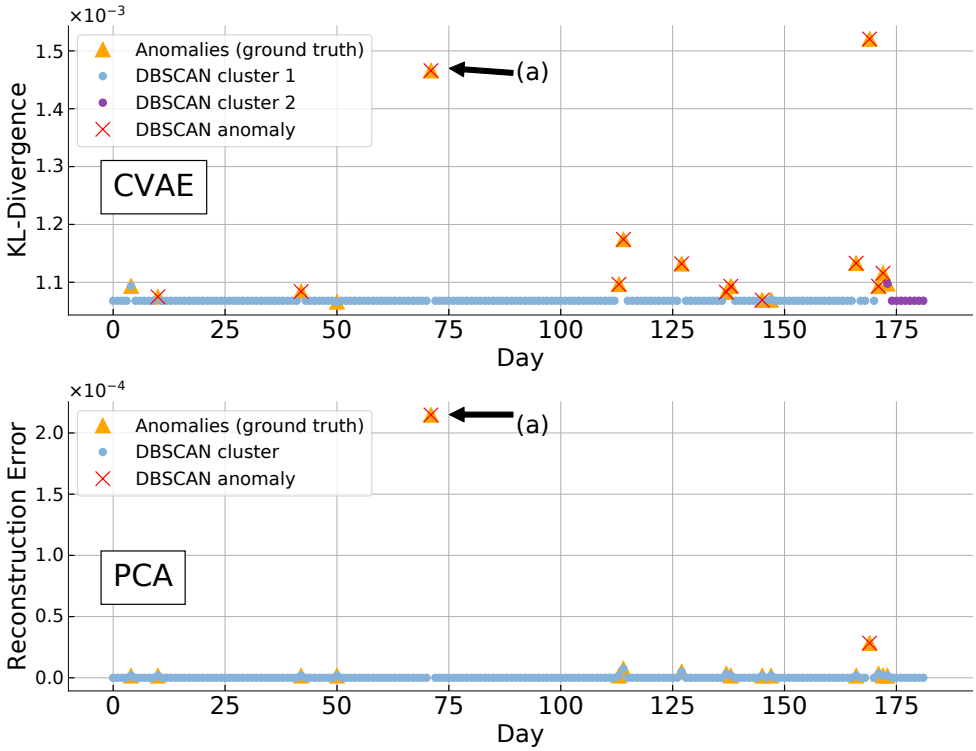

Fig. 5.9: Robustness of CVAE against large outliers

■Robustness against outliers: KLD is robust against the intensity (i.e., number of log appearances) of the original data, as shown in Fig 5.3. On the other hand, we confirm that PCA's reconstruction errors (not shown in the figure) are also biased to the intensity, similar to the case with MLH.

Fig 5.9 (MPLS path down event) shows an example of CVAE's sensitivity to a small number of log appearances. The top graph shows KLD with CVAE, and the bottom graph shows the reconstruction errors with PCA. The solid blue circles are normal points and the crosses are anomalies based on DBSCAN clustering. Note that anomalies in the top graph are clustering results after merging ten trials, but the KLD values are a result of one trial. These data have one large outlier (around day 17; label (a)) which has a larger number of log appearances than other days. As shown in the figure, PCA is affected by this outlier, and many anomalous points are not detected. With CVAE, however, the KLD values are relatively spread compared with reconstruction errors, and DBSCAN detects other anomalies. However, we also confirm that less than 1% of the data have high KLD values ($> 10^{-4}$), as shown in Fig 5.3. It is still possible that the clustering process may miss small KLD value changes due to these high values. We will work to improve this issue by tuning the clustering process as future work.

Therefore, CVAE exhibits better robustness to outliers and noises than PCA.

5.8 Case Study: BGP state change

We now discuss anomaly examples detected from our proposed method in Fig 5.10. In this figure, each mark shows the KLD of the BGP state change logs from a router per day. Almost all days have a baseline ($\approx 2.35 \times 10^{-2}$). Checking the raw logs around the 25th day (a red area labeled (a) in the figure), we find that a connection with one particular AS suddenly became unstable.

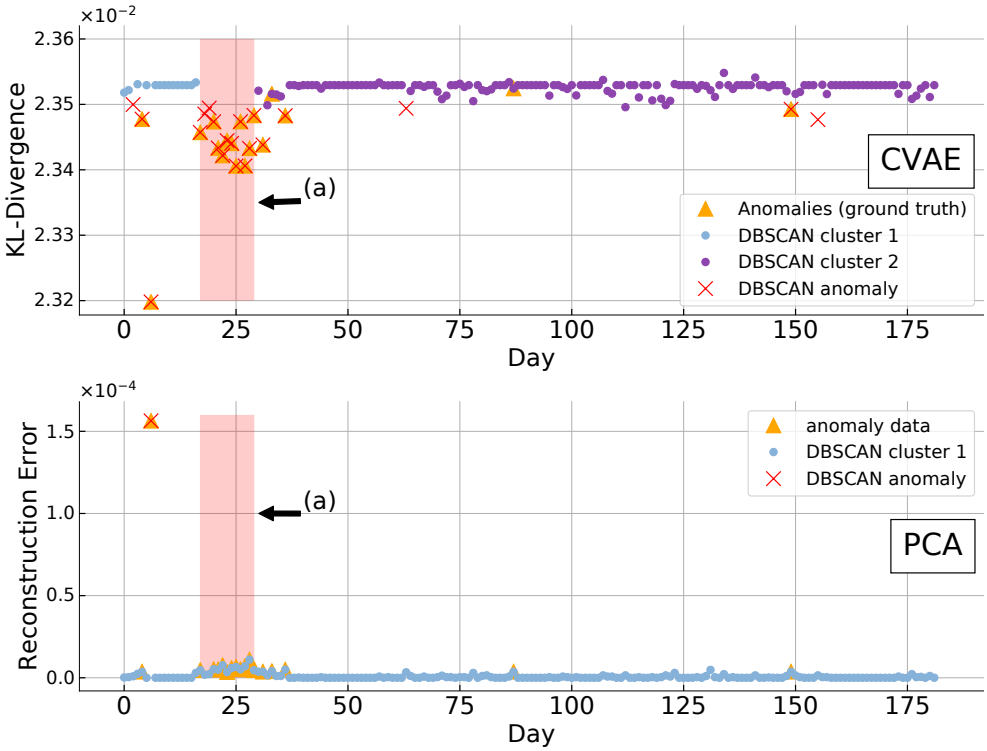

Fig. 5.10: Case study: BGP neighbor state change logs

These failures had been also reported in a trouble ticket. Thus, they are useful information for network operators or troubleshooting. As shown in the bottom figure, PCA fails to detect most of these anomalies due to a larger outlier. CVAE, on the other hand, successfully discovered them.

5.9 Summary

We proposed a log analysis method based on the latent variable model to detect network system anomalies. We used Conditional Variational Autoencoder (CVAE) and applied a clustering algorithm to log time series with KL-divergence of the latent variable distribution. We confirmed that this method works well for highly sparse and diverse time series through comparison with Kleinberg's burst detection and a traditional PCA-based method. Our method achieved 14.5% higher recall and 3% higher precision than a PCA-based method. We also evaluated performance dependency of different training data size and concluded 3 month or larger size of training data is sufficient for our proposed method. In addition, our case study showed that some detected anomalies are helpful in finding network troubles. Towards the deployment of our method to real network management, we will work on addressing how to update the trained model with upcoming syslog data.

Chapter 6

Discussion

6.1 Limitations

The common limitation in burstiness analysis and latent variable based anomaly detection is coverage of the type of dataset. Since we only use SINET4 dataset in this research, we will evaluate our methods in other network system as a future work.

Considering practical use of latent variable based anomaly detection, there are three limitations. First, even though our method does not require data-specific preprocessing, we have to tune hyper parameters of CVAE. We find the parameters are different according to the scale of dataset. Second, our method does not work on real time anomaly detection because the bin size of input data is 1 day long. Thus, our method can be used as a daily operation. Third, our method cannot handle whole dataset became of the dimension of input data. In this work, we use a part of SINET4 dataset in anomaly detection. It has 1,440 dimension as a time series and 120 dimension as a conditional label. The whole dataset includes about 9,000 dimension as a conditional label and we confirmed the loss function in training phase was not converged in the whole dataset. Thus, we have to reduce the dimension of conditional labels. One way to reduce conditional label is using multi hot label; for example, using combination of device ID and template ID. We confirmed this approach also worked well through a experimental results.

6.2 Findings and contributions

In this research, we focus on automatic system log analysis in a network system and conducted burstiness analysis and latent variable based anomaly detection.

As a result of burstiness and causality analysis, we confirm 92.5% of log bursts are trivial and our preprocessing can remove these trivial bursts. To combine burst detection results, causal inference results and other metrics such as burst co-occurrence ratio and dynamic time warping distance, we reduce a large amount of trivial bursts, and extract meaningful information from log messages. We find that 7.5% of co-occurred bursts have true causality. Our similarity analysis separates whether the causality of co-burst comes from one network activity or multiple ones. In the device-based burst detection, we find 3,735 bursts that are only found by this multi variate analysis. Also we find some causal bursts which are not found in the existing causal inference results.

Next, we proposed latent variable based anomaly detection in network logs. In order to detect anomaly without any data-specific preprocessing or definitions of anomaly, we employ latent variables. CVAE learns diverse trend of log time series and represents time series characteristics by latent variables. Our proposed method can detect not only anomalous appearance of logs such as burstiness, but also anomalous disappearance of logs such as lack of periodicity. Through comparison with PCA based anomaly detection, which is one of traditional unsupervised methods, our method achieves 14.5% higher recall and 3% higher precision. This method is adaptive to the

change of network behavior because we only use log time series and log template information, not any data-specific knowledge.

6.3 Future work

Looking back at the open problems mentioned in §1, we tackled on the two problems through this work. As shown in §2.3, there are some existing methods which also focus on unsupervised detection. However, they worked in application layer logs, which are more specific and less diverse than network layer logs, or not focused on anomaly detection but knowledge mining. In this work, we showed our latent variable based anomaly detection can handle adaptivity problem even in network logs. Considering the gap between statistical anomaly and real problem, our approach is still not enough because detected anomaly is statistical result. Thus, we have to implicate or investigate the anomaly in order to find why or how the anomaly was occurred. We consider our burstiness and causality analysis framework can reduce a part of this gap. In burstiness and causality analysis, we used results of causality analysis as a knowledge. Similarly, we can utilize the results of causal analysis to implicate detected anomalies by our proposed method. We consider that semantic information of logs will be also useful as a knowledge.

Now, the adaptivity problem is being solved and therefore we should focus on the gap problem between statistical anomaly and real problems for more practical analysis in network management.

Chapter 7

Conclusion

In this study, we worked on the system log anomaly detection towards supporting large scale network operation. We consider that there are three large problems about log analysis: difficulties in analyzing logs, usefulness in network management and adaptivity to the change of network system behavior. We tackled on the second problem in burstiness and causality analysis.

In this analysis, we first simply detected burstiness in network logs and analyze usefulness based on heuristic and the results of log causality analysis. As a result, we found 99% of detected bursts were trivial and our preprocessing successfully removed them. In addition, we found 7.5% of co-occurred bursts had true causality and our similarity analysis separated whether the causality of co-burst comes from one network activity or multiple ones. In the device-based burst detection, we find 3,735 bursts that are only found by this multi variate analysis. Also we find some causal bursts which are not found in the existing causal inference results.

Next, we focused on the third problem and proposed latent variable based log anomaly detection. This method can detect anomaly which is deviated behavior from normal state without any specific preprocessing. Our evaluation showed that our method achieved 14.5% higher recall and 3% higher precision than traditional PCA based anomaly detection. Our case study showed that some detected anomalies are helpful in finding network troubles such as BGP unstable connections.

Through this work, we tackled on the two problems about log analysis: gap between statistical anomaly and real problem, adaptivity to the change of network system behavior. Our latent variable based anomaly detection could handle the adaptivity in network system. However, the gap between statistical anomaly and real problem is still remained although our burstiness and causality analysis partially solved this problem. We will work on applying burstiness and causality analysis framework into our latent variable based anomaly detection in order to fill the gap between numerical anomaly and real system anomaly.

外部発表論文

Journal

(A) K. Otomo, S. Kobayashi, K. Fukuda, H. Esaki, "Latent Variable based Anomaly Detection in Network System Logs," IEICE Transactions on Information & Systems Special Section on Log Data Usage Technology and Office Information Systems (submitted).

(B) S. Kobayashi, K. Otomo, K. Fukuda, H. Esaki, "Mining Causality of Network Events in Log Data," IEEE Transactions on Network and Service Management, vol. 15, pp. 53-67, 2018 (reviewed).

Conference (non-reviewed)

(C) 大友 一樹, 小林 論, 福田 健介, 江崎 浩, "システムログ異常予測に向けたバースト性と因果関係の解析", The Eighteenth Workshop on Internet Technology (WIT2017), 愛媛, 2017 年 6 月.

(D) 小林 論, 大友 一樹, 福田 健介, "ネットワーク構成情報を考慮したネットワークログ因果解析の検討", インターネットアーキテクチャ研究会, 広島, 2018 年 12 月.

Conference (reviewed)

(E) K. Otomo, S. Kobayashi, K. Fukuda, H. Esaki, "Finding Anomalies in Network System Logs with Latent Variables," ACM SIGCOMM 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (BIGDAMA2018), pp. 8-14, Budapest, Hungary, August 2018.

(F) K. Otomo, S. Kobayashi, K. Fukuda, H. Esaki, "Analyzing Burstiness and Causality of System logs", ACM SIGCOMM CoNEXT 2017 Student Workshop, Seoul, South Korea, December 2017 (Poster session).

(G) K. Otomo, S. Kobayashi, K. Fukuda, H. Esaki, "An Analysis of Burstiness and Causality of System Logs", Asian Internet Engineering Conference (AINTEC 2017), Bangkok, Thailand, November 2017.

References

[1] J. Davin. The Syslog Protocol. RFC 1157, 1990.

[2] R. Gerhards. A Simple Network Management Protocol (SNMP). RFC 5244, 2009.

[3] Elisabeth Baseman, Sean Blanchard, and Email Zongzelimyuntedu. Relational Synthesis of Text and Numeric Data for Anomaly Detection on Computing System Logs. *in Proc. IEEE ICMLA'16*, 1:2–5, 2016.

[4] Jiang Zhong, Weili Guo, and Zhenhua Wang. Study on network failure prediction based on alarm logs. *In Proc. ICBDSC'16*, (2015):23–29, 2016.

[5] Melody Moh, Santhosh Pininti, Sindhusha Doddapaneni, and Teng-Sheng Moh. Detecting Web Attacks Using Multi-stage Log Analysis. *in Proc. IEEE IACC'16*, pages 733–738, 2016.

[6] M Shatnawi and M Hefeeda. Real-time failure prediction in online services. *in Proc. IEEE INFOCOM'15*, pages 1391–1399, 2015.

[7] Siyang Lu, Bingbing Rao, Xiang Wei, Byungchul Tak, Long Wang, and Liqiang Wang. Log-based Abnormal Task Detection and Root Cause Analysis for Spark. *In Proc. IEEE ICWS*, 2017.

[8] Satoru Kobayashi, Kazuki Otomo, Kensuke Fukuda, and Hiroshi Esaki. Mining causality of network events in log data. *IEEE TNSM*, 15(1):53–67, 2018.

[9] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience Report : System Log Analysis for Anomaly Detection. *in Proc. IEEE ISSRE'16*, pages 207–218, 2016.

[10] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. *in Proc. IPOM'03*, pages 119–126, 2003.

[11] Risto Vaarandi. Mining event logs with SLCT and LogHound. *In Proc. IEEE IM'08*, pages 1071–1074, 2008.

[12] Adetokunbo A. O. Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. Clustering event logs using iterative partitioning. *In Proc. ACM SIGKDD'09*, page 1255, 2009.

[13] Adetokunbo Makanju, A Nur Zincir-heywood, and Evangelos E Milios. Message Type Extraction Based Alert Detection in System Logs. *Cs.Dal.Ca*, pages 1–16, 2009.

[14] Adetokunbo Makanju, a. Nur Zincir-Heywood, and Evangelos E. Milios. Storage and retrieval of system log events using a structured schema based on message type transformation. *in Proc. ACM SAC'11*, pages 528–533, 2011.

[15] Masayoshi Mizutani. Incremental mining of system log format. *in Proc. SCC'13*, pages 595–602, 2013.

[16] P. He, J. Zhu, Z. Zheng, and M. R. Lyu. Drain: An online log parsing approach with fixed depth tree. pages 33–40, June 2017.

[17] Qiang Fu, Jian Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. *In Proc IEEE ICDM'09*, pages 149–158, 2009.

[18] Liang Tang, Tao Li, and Chang-Shing Perng. Logsig: Generating system events from raw textual logs. *In Proc. ACM CIKM'11*, pages 785–794, 2011.

[19] Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. Towards an NLP-based log template generation algorithm for system log analysis. *in Proc. CFI'14*, pages 1–4, 2014.

[20] Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael I Jordan, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting Large-Scale System Problems by Mining Console Logs. *In Proc. ACM SIGOPS'09*, pages 117–131, 2009.

[21] Edward Chuah, Shyh Hao Kuo, Paul Hiew, William Chandra Tjhi, Gary Lee, John Hammond, Marek T. Michalewicz, Terence Hung, and James C. Browne. Diagnosing the root-causes of failures from cluster log files. *in Proc. HiPC'10*, pages 1–10, 2010.

[22] Tatsuaki Kimura, Akio Watanabe, Tsuyoshi Toyono, and Keisuke Ishibashi. Proactive Failure Detection Learning Generation Patterns of Large-scale Network Logs. *in Proc. CNSM'15*, pages 8–14, 2015.

[23] R. Vinayakumar, K. P. Soman, and P. Poornachandran. Long short-term memory based operation log anomaly detection. *In Proc ACM ICACCI'17*, pages 236–242, Sept 2017.

[24] M. Nadeem, V. Nigam, D. Anagnostopoulos, and P. Carretas. Automating network error detection using long-short term memory networks. *arXiv*, 2018.

[25] W. Meng, Y. Liu, S. Zhang, D. Pei, H. Dong, L. Song, and X. Luo. Device-agnostic log anomaly classification with partial labels. *arXiv*, 2018.

[26] T. Tan, S. Gao, W. Yang, Y. Song, and C. Lin. Two new term weighting methods for router syslogs anomaly detection. *In Proc IEEE HPCC'16*, pages 1454–1460, Dec 2016.

[27] Du. Min, Li. Feifei, Zheng. Guineng, and Srikumar. Vivek. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. *In Proc ACM CCS'17*, pages 1285–1298, 2017.

[28] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *In Proc. ACM SIGCOMM'04*, 34:219, 2004.

[29] Z. Zheng, Z. Lan, B. H. Park, and A. Geist. System log pre-processing to improve failure prediction. *In Proc IEEE DSN'09*, pages 572–577, 2009.

[30] 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 network events. *In Proc IEEE INFOCOM'14*, pages 610–618, 2014.

[31] T. Hacker, R. Pais, and C. Rong. A markov random field based approach for analyzing supercomputer system logs. *IEEE Transactions on Cloud Computing*, pages 1–1, 2017.

[32] Peter Spirtes and Clark Glymour. An Algorithm for Fast Recovery of Sparse Causal Graphs. *Social Science Computer Review*, 9(1):62–72, 1991.

[33] Markus Kalisch and Peter Buehlmann. Estimating high-dimensional directed acyclic graphs with the PC-algorithm. *Journal of Machine Learning Research*, 8:613–636, 2005.

[34] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. CauseInfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. *in Proc. IEEE INFOCOM'14*, pages 1887–1895, 2014.

[35] Satoru Kobayashi. Mining causes of network events in log data with causal inference. *in Proc. IEEE IM'17*, pages 45–53, 2017.

[36] Shigeo Urushidani, Michihiro Aoki, Kensuke Fukuda, Shunji Abe, Motonori Nakamura, Michihiro Koibuchi, Yusheng Ji, and Shigeki Yamada. Highly available network design and resource management of SINET4. *Telecommunication Systems*, 56(1):33–47, 2014.

[37] Jon Kleinberg. Bursty and Hierarchichal structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.

[38] Donald Berndt and James Clifford. Using dynamic time warping to find patterns in time series. *in Proc. ACM KDD'94*, 398:359–370, 1994.

[39] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv*, pages 1–14, 2013.

[40] Diederik P. Kingma, Danilo J. Rezende, Shakir Mohamed, and Max Welling. Semi-Supervised Learning with Deep Generative Models. *arXiv*, pages 1–9, 2014.

[41] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *In Proc. ACM KDD'96*, pages 226–231, 1996.

[42] Kazuki Otomo, Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. An Analysis of Burstiness and Causality of System Logs. *In Proc. AINTEC'17*, pages 16–23, 2017.

Acknowledgements

I would first like to express my profound gratitude to Professor Hiroshi Esaki for providing me a precious opportunity of study as a master student in his laboratory. I would like to thank appreciation to Associate Professor Hideya Ochiai, Dr.Tsukada. They taught me a lot of things about how to progress research and how to compose thesis as well as how to work with colleagues in laboratory. I especially would like to express my deepest appreciation to Associate Professor Kensuke Fukuda at National Institute of Informatics for leading my work to success and giving me a lot of beneficial advice. I also deeply grateful to Dr.Satoru Kobayashi to discuss my research and giving me a lot of advice. I would like to thank all of our lab members, especially Yutaro Nomura, Masahiro Kitazawa, Toki Suga, Genta Imai and Daichi Koda for being good friends of mine, and making my lab life joyful. Lastly, my heartfelt appreciation goes to my family for their moral support and warm encouragements.