LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs

Weibin Meng1,5, Ying Liu1,5, Yichen Zhu2, Shenglin Zhang3*, Dan Pei1,5, Yuqing Liu3, Yihao Chen1,5, Ruizhi Zhang4, Shimin Tao4, Pei Sun4 and Rong Zhou4

1Tsinghua University
2University of Toronto
3Nankai University
4Huawei
5Beijing National Research Center for Information Science and Technology (BNRist)
mwb16@mails.tsinghua.edu.cn, liuying@ccer.edu.cn, k.zhu@mail.utoronto.ca, zhangsl@nankai.edu.cn, peidan@tsinghua.edu.cn, lyq@mail.nankai.edu.cn, chenyiha17@mails.tsinghua.edu.cn, {zhangruizhi, taoshimin, sunpei.sun, jce.zhourong}@huawei.com

1 Introduction

Today’s large-scale services are becoming increasingly more agile and complicated. A single service anomaly can impact million of users’ experience [Bu et al., 2018; Zhang et al., 2015; Ma et al., 2018]. Accurate and timely anomaly detection can help operators quickly mitigate losses [Zhang et al., 2018b], which is crucial for these services. Large-scale services usually generate logs, which describe a vast range of events observed by them, to record system states at runtime. Logs are one of the most valuable data sources for anomaly detection [Satpathi et al., 2018; Lin et al., 2016; Du et al., 2017; Khatuya et al., 2018; Nandi et al., 2016; He et al., 2018; Meng et al., 2018; Zhang et al., 2018a].

*Shenglin Zhang is the corresponding author.

Abstract

Recording runtime status via logs is common for almost computer system, and detecting anomalies in logs is crucial for timely identifying malfunctions of systems. However, manually detecting anomalies for logs is time-consuming, error-prone, and infeasible. Existing automatic log anomaly detection approaches, using indexes rather than semantics of log templates, tend to cause false alarms. In this work, we propose LogAnomaly, a framework to model a log stream as a natural language sequence. Empowered by template2vec, a novel, simple yet effective method to extract the semantic information hidden in log templates, LogAnomaly can detect both sequential and quantitative log anomalies simultaneously, which has not been done by any previous work. Moreover, LogAnomaly can avoid the false alarms caused by the newly appearing log templates between periodic model retrainings. Our evaluation on two public production log datasets show that LogAnomaly outperforms existing log-based anomaly detection methods.

A large-scale service and its underlying machines are often implemented/maintained by hundreds of developers/operators. Usually a developer/operator has incomplete information of the overall system, and tend to determine anomalous logs from a local perspective and thus is error-prone. In addition, manual detection of anomalous logs is becoming infeasible due to the explosion of logs. Keywords (e.g., “fail”) matching and regular expressions, detecting single anomalous logs based on explicit keywords or structural features, prevent a large portion of log anomalies from being detected. These anomalies can only be inferred based on their log sequences which contains multiple logs violating regular rules. For example, the first four logs in Figure 1 show two normal link flaps. If we apply keyword matching to detect log anomalies, both L1 and L2, which contain the keyword “down”, will trigger false alarms. However, it is actually a normal event because the switch automatically recovers quickly as demonstrated in L4. Consequently, an automatic anomaly detection method according to log sequences is needed.

Generally, there are two categories of log sequence anomalies: sequential and quantitative anomalies. Programs are typically executed according to fixed flows, and logs are a sequence of events produced by these executions. We say a
sequential anomaly occurs if a log sequence deviates from normal patterns of program flows. Meanwhile, program execution has some constant linear relationships, which can be captured by the quantitative relationships of logs, should always hold true under different workloads. We say a quantitative anomaly occurs if these relationships are broken for a collection of logs. Existing automatic anomalous log sequence detection approaches can be broadly classified into two categories: log message counter based approaches (e.g., PCA [Xu et al., 2009], Invariant Mining [Lou et al., 2010], LogClustering [Lin et al., 2016]) to capture quantitative anomalies, and deep learning based approaches (e.g., DeepLog [Du et al., 2017]) to learn sequential patterns from log sequences. These methods all take log template indexes as input, which can often induce false alarms. For example, as shown in Figure 1, words with underlines are variables, and the remaining parts are templates, each of which is usually indexed by a numerical identifier. Suppose that the above methods have been trained based on the normal log sequence, i.e., \( L_3 \) to \( L_4 \). When a system generates \( L_5 \) (the templates of \( L_2 \) and \( L_6 \) are very similar but different, and \( L_1 \)) \( L_3 \) \( L_4 \) has the same template with \( L_5 \) \( L_7 \) \( L_8 \), the above methods will mistakenly think that the log sequence of \( L_5 \) to \( L_8 \) is anomalous, based on the observation that \( L_2 \) and \( L_6 \) have different template indexes. Above all, the anomalous log sequence detection problem faces the following three challenges.

1. Valuable information could be lost if only log template indexes are used, because they cannot reveal the semantic relations of logs. For example, some templates are similar in semantics but different in template indexed, and ignoring this similarity can induce false alarms.

2. Services can generate new log templates between two adjacent periodic re-trainings, and existing approaches cannot address this problem. For instance, manual feedback on large number of new log templates (as is done by [Du et al., 2017]) is infeasible in practice.

3. Existing methods cannot detect sequential and quantitative anomalies simultaneously.

We propose LogAnomaly, a unified data-driven deep-learning framework for anomaly detection on unstructured log streams. The core idea of LogAnomaly is that most system logs are semi-structured texts “print”-ed by certain procedures of systems, and the intuitions and methods in natural language processing can be applied or improved for log anomaly detection. LogAnomaly tackles the above challenges as follows.

1. Inspired by word embedding, we design a simple yet effective template representation method, template2Vec, to accurately extract the semantic and syntax information from log templates. Clearly, template2Vec captures not only word context, but also semantic information including synonyms and antonyms (in our scenario, logs with antonyms usually indicate different events). To the best of our knowledge, this is the first anomaly detection framework considering semantic information of logs.

2. We design a mechanism to merge new templates without operators’ feedback between two adjacent trainings.

![Figure 2: Logs and templates](image-url)

3. We propose LogAnomaly, an end-to-end framework using LSTM network to automatically detect sequential and quantitative anomalies simultaneously.

We evaluate LogAnomaly on two benchmark datasets in log analysis scenarios, the HDFS dataset [Xu et al., 2009] and the BGL dataset [Oliner and Stearley, 2007]. Our results show that LogAnomaly outperforms state-of-the-art log-based anomaly detection methods.

## 2 Background

Large-scale systems usually record system runtime states using logs, generated using the “print” function with a string template (normal text in Figure 1) and detailed information as parameters (the underscored text in Figure 1). Typically, these logs have to be properly parsed before they can be effectively used for anomaly detection [He et al., 2016]. A common approach to parse logs is to mine and extract templates from historical logs and then match logs to templates [Zhang et al., 2017; Messaoudi et al., 2018]. As shown in Figure 2, logs \( L_3 \) and \( L_6 \) are similar and their templates are both \( T_3 \), i.e., “Interface * changed state to up”, which sketches out the event that \( L_3 \) and \( L_6 \) represent. The remaining parts of logs \( L_3 \) and \( L_6 \) (i.e., “ae3” and “ae1”), respectively, are the runtime parameters. Log parsing is beyond the scope of this paper, thus in Section 3.4, we simply adopt FT-Tree [Zhang et al., 2017], which is one of the state-of-art log parsing methods, achieving high accuracy in template extraction and being incrementally retrainable when new types of logs emerge.

There are two dimensions to classify existing log anomaly detection approaches: 1) detecting sequential anomalies or quantitative anomalies; 2) supervised or unsupervised. For those approaches designed to detect quantitative anomalies, a time or session window is defined, and then the count of each template index (regardless of sequence) within the window is used as the basis for anomaly detection. Supervised approaches can be used to detect anomalies using various algorithms: logistic regression and decision tree (in [He et al., 2016]) and SVM (in [Liang et al., 2007]). Among unsupervised approaches that detect quantitative anomalies, [Xu et al., 2009] uses PCA; [Lin et al., 2016] (LogCluster) uses clustering; [Lou et al., 2010] (Invariants Mining) detects whether some mined invariants (e.g., \( \text{count}(T_1) = \text{count}(T_3) \)) hold true within the window.

[Zhang et al., 2016] and DeepLog [Du et al., 2017] are
3 Design of LogAnomaly

3.1 Overview

The objective of LogAnomaly is to, in an unsupervised way, automatically and accurately detect both sequential and quantitative log anomalies in real time.

After extensive investigations into real-world logs, we have the following two observations: (1) A log template, which is predefined using the “print” function by developers, typically characterizes the event that occurs in the system, with its text representing the semantic information of the event. (2) Program execution flows in a service system usually have different. For example, the word “down” in template T1 and the word “up” in template T2 in Figure 2, which are antonyms, have the same context. Obviously, these two templates express the opposite meaning, and they should have very different template vectors. Therefore, we try to capture the semantic information when constructing vectors to represent templates.

We propose a novel word representation method, template2Vec, which is based on synonyms and antonyms, to effectively represent the words in templates distributedly. It creates vectors that are distributed representations of word features, such as the context of individual words. However, it does not capture the semantic information of log sequence. An anomaly is declared if the actual log is unexpected according to the prediction.

The design of our proposed LogAnomaly is shown in Figure 3. In the offline learning component, LogAnomaly first leverages FT-Tree to extract templates from historical logs and then match the historical logs to these templates. Each template is a set of words in semi-structured text. Inspired by word2Vec [Mikolov et al., 2013], we propose a novel method, template2Vec, which distributedly represents templates. It effectively converts the words in templates into word embedding vectors and calculates template vectors by combining word vectors. This way, a log sequence is converted to a template vector sequence. In the end, we extract the sequential and quantitative features from this log sequence with an LSTM model. The offline training is conducted periodically, e.g., weekly, so that the newly appeared log templates can be periodically incorporated into the newly learned offline model.

In the online detection component, we first determine, for a realtime log, whether it can be matched to an existing template. If yes, we then convert it to a template vector. Otherwise, we will “approximate” the “temporary” template vector to an existing one based on the similarity of template vectors. Consequently, each realtime log is matched to a template vector, and realtime logs are converted to (template) vector sequences. Based on the trained LSTM model in the offline learning component, LogAnomaly determines whether a log sequence is anomalous.

3.2 Template2Vec

Word2Vec is a popular distributed representation method for words. It creates vectors that are distributed representations of word features, such as the context of individual words. However, it does not capture the semantic information of words such as synonym and antonym. The words that have similar or the same context in two log templates, in many cases, can be antonyms, which makes the two templates very different. For example, the word “down” in template T1 and the word “up” in template T2 in Figure 2, which are antonyms, have the same context. Obviously, these two templates express the opposite meaning, and they should have very different template vectors. Therefore, we try to capture the semantic information when constructing vectors to represent templates.

We propose a novel word representation method, template2Vec, which is based on synonyms and antonyms, to effectively represent the words in templates distributedly. As shown in Figure 4, template2Vec includes three steps in the offline learning stage: (1) Construct the set of synonyms and antonyms. As shown in Table 1, universal synonyms and antonyms can be found in WordNet [Miller, 1995], which is a lexical database for English. Some domain specific synonyms and antonyms, however, have to be added by operators based on domain knowledge. Therefore, we first search synonyms and antonyms of the words in templates in WordNet. After that, operators can update synonyms and antonyms. (2) Generate word vectors. We apply dLCE [Nguyen et al., 2016], which is a distributional lexical-contrast embedding model, to
generate word vectors that distributedly represent the words in templates. (3) Calculate template vectors. For a given template, we calculate its template vector, which is the weighted average of the word vectors of the words in the template, to represent the distribution of the template. In online detection stage, it will match a "temporary" template vector to an existing one for new types of logs. Therefore, template2Vec combines operators’ domain knowledge and dLCE model in order to accurately generate template vectors.

### 3.3 Log Anomaly Detection

As aforementioned, programs, which generate logs using the “print” function, are usually executed according to fixed flows. Therefore, normal logs naturally have some sequential patterns. In other words, for a given template (vector) sequence, its next template is predictable if no anomaly occurs. Let $\Omega = \{v_1, v_2, ..., v_n\}$ be the whole set of distinct template vectors. The sequence for detection is a sliding window of the $w$ most recent template vectors. For a log sequence $S = \{s_1, s_2, ..., s_m\}$, suppose that $S_j = \{s_j, s_{j+1}, ..., s_{j+w-1}\}$ is one of its subsequences. Figure 5 (a) shows the sequential pattern of the templates shown in Figure 2. For example, the next vector of $[v_1, v_2, v_3]$ is likely to be $v_1$. Because LSTM is a popular recurrent neural network architecture that has been proved to robustly predict for sequences of data [Du et al., 2017; Zhang et al., 2016], we apply LSTM to learn the sequential pattern of logs. Then the template vector sequence of $S_j$ is $V_j = \{v(s_{ij}), v(s_{ij+1}), ..., v(s_{ij+w-1})\}$, where $v(s_i)$ is the template vector of $s_i$, and $v(s_i) \in \Omega$. Therefore, $V_j$ is input into the LSTM model for $S_j$ in the training stage.

In addition to sequential patterns, a template (vector) sequence also has quantitative patterns. Usually, a normal program execution has some invariants, and some quantitative relationships always hold in logs under different inputs and workloads. For example, each opened file will be finally closed at some stage. Therefore, the number of the logs indicating “open a file” should be equal to that of the logs showing “close a file” in a normal condition. These quantitative relationships in logs can capture the normal program execution behavior. If a new log breaks certain invariants, we can determine that an anomaly occurs during the system execution. Consequently, we learn the quantitative pattern of logs as follows. For a log message $s_i \in S_j$, we calculate the count vector of the log sequence $(s_{i-w+1}, s_{i-w+2}, ..., s_i)$, which is denoted as $C_j = (c_j(v_1), c_j(v_2), ..., c_j(v_n))$, where $c_j(v_k)$ is the number of $v_k$ in the template vector sequence $(v(s_{i-w+1}), v(s_{i-w+2}), ..., v(s_i))$, and $v_k \in V$. Finally, $C_j, C_{j+1}, ..., C_{j+w-1}$ is input into LSTM to learn the quantitative pattern of $S_j$.

In general, there are several branches in the fixed flows of program execution. For instance, after a file is opened, it can be read, written, or closed. Therefore, based on the sequential and quantitative patterns of program execution and logs, there are several possibilities for the next template vector after a template (vector) sequence. For a log sequence, we sort the possible next template vector based on their probabilities, which is learned according to the LSTM model. If the observed next template vector is included in the top $k$ candidates (or similar enough with them), we regard it as normal.

LogAnomaly learns the inherently sequential and quantitative patterns of logs by learning the high dimensional dependencies of template sequences. The combination of the sequential and quantitative patterns, we believe, can improve the accuracy, which is demonstrated in Section 4.2.

### 3.4 Template Approximation Between Two Consecutive Trainings

In real-time systems, systems may generate new log templates online, but existing approaches cannot handle newly generated template indexes. Manual feedback on large number of new log templates (as is done by [Du et al., 2017]) is infeasible in practice. Although periodic retraining can alleviate the problems, we still have to deal with the new log templates that appear between two consecutive offline trainings. Our approach is as follows.

In the online detection stage, for a log that cannot match any existing template, we first apply FT-Tree to extract a “temporary” template from the new log, and calculate its template vector. After that, we match this “temporary” template vector to an existing one based on the similarity among template vectors. The intuition is that majority of the “new” template is just minor variants of existing templates, instead of brand new ones. Note that the template approximation is aimed to deal with the newly appeared log templates between two consecutive offline trainings. This is a better approach than the labor-intensive manual feedback by operators (as is in [Du et al., 2017])

Therefore, the challenge imposed by new types of logs is addressed without operators’ intervention. In this way, the number of distinct template vectors remains constant, even though new types of logs can emerge.

### 4 Evaluation

#### 4.1 Experiment Setting

**Datasets**

We conduct experiments over the BGL dataset [Oliner and Stearley, 2007] and the HDFS dataset [Xu et al., 2009], as listed in Table 2. The detailed information of the two datasets are described as follows:
### Datasets Duration # of logs # of anomalies

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Duration</th>
<th># of logs</th>
<th># of anomalies</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td>38.7 hours</td>
<td>11,175,629</td>
<td>16,838(blocks)</td>
</tr>
<tr>
<td>BGL</td>
<td>7 months</td>
<td>4,747,963</td>
<td>348,460(logs)</td>
</tr>
</tbody>
</table>

Table 2: Detail of the datasets

(1) **BGL**: The BGL dataset contains 4,747,963 logs. Each BGL log was manually labeled as either anomalous or normal, and 348,460 logs were labeled as anomalous. The BGL dataset was generated by the Blue Gene/L supercomputer, which consisted of 128K processors and was deployed at Lawrence Livermore National Laboratory (LLNL).

(2) **HDFS**: The HDFS dataset consists of 11,175,629 logs collected from more than 200 Amazon EC2 nodes. A program execution in the HDFS system, e.g., writing a file and then closing it, usually involves a block of logs. There are 575,061 blocks of logs in the dataset, among which 16,838 blocks were labeled as anomalous by Hadoop domain experts [Da et al., 2017].

In the following experiments, from either dataset, we leverage the front 80% (according to the timestamps of logs) as the training data, and the rest 20% as the testing data. Moreover, because both of the above datasets were manually labeled, we take these labels as the groundtruth for evaluation.

### Baselines

We compare LogAnomaly with four unsupervised baseline methods, i.e., LogCluster [Lin et al., 2016], PCA [Xu et al., 2009], Invariant Mining (IM) [Lou et al., 2010], and Deeplog [Du et al., 2017] (see Section 2 for more details). The parameters of these methods are all set best for accuracy.

### Evaluation Metrics

Anomaly detection is a binary classification problem. A classification method’s capability is usually assessed by precision, recall, and F1 score. We label its outcome as a TP, TN, FP, and FN. TP are the anomalous logs (or blocks for the HDFS dataset) that are accurately determined as such by the method. TN are normal logs (or blocks for the HDFS dataset) that are accurately determined. If the method determines a log (or a block for the HDFS dataset) as anomalous, but in fact it is normal, we label the outcome as a FP. The rest are FN.

\[
\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN},
\]

F1 score = \(2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}\).

### Experimental Setup

We conduct all the experiments on a Linux server with Intel Xeon 2.40 GHz CPU and 64G memory. FT-Tree [Zhang et al., 2017] is leveraged to parse logs. We implement LogAnomaly and DeepLog with Python 3.6 and Keras 2.1. As for Log Clustering, PCA and Invariant Mining, we use a popular open-source toolkit implemented in [He et al., 2016].

### 4.2 Evaluation of The Overall Performance

In this section, we compare LogAnomaly with the four baseline methods on the two datasets. The LogAnomaly in our experiments has two LSTM layers with 128 neurons, and the size (step) of window is 20 (1). Figure 6 and Figure 7 respectively show the comparison results of LogAnomaly and the four baseline methods on the BGL dataset and the HDFS dataset. Overall, LogAnomaly achieves the best accuracy among the five methods, having an averaged F1 score of 0.96 on the BGL dataset, and 0.95 on the HDFS dataset. Both PCA and LogCluster, however, have low F1 scores (≤ 0.80) on the two datasets. Both Invariants Mining and Deeplog have recalls of 0.99 and 0.96 on the BGL dataset, respectively. However, these high recalls are at cost of low precisions. For instance, on the BGL dataset, Invariants Mining and Deeplog respectively decrease precisions by 0.14 and 0.07 compared to LogAnomaly, resulting that they generate 5.67 and 3.33 times false alarms, respectively. Generally, large service systems produce tens of millions of logs everyday. If a log anomaly detection method generates too many false alarms everyday, it will consume operators a large amount of unnecessary work. Both Invariants Mining and Deeplog leverage the indexes of log templates, which ignore the semantic information of templates, to learn the anomalous and normal patterns. However, different templates having different indexes, e.g., T1 and T2 in Figure 2, can share common semantic information. Ignoring the above common semantic information results in that both Invariants Mining and Deeplog generate more false alarms than LogAnomaly. On the contrary, LogAnomaly, which employs template2Vec to learn the semantic information of log templates based on synonyms and antonyms, effectively avoids these false alarms. That is because comparing the similarity of template vectors, which LogAnomaly falls into, is tolerant to the small differences of log templates, than comparing the similarity of template indexes, where Invariants Mining and Deeplog fail.

In this work, for the first time, we learn both the quantitative pattern and the sequential pattern of logs for anomaly detection. To demonstrate the benefits of the combination of these two patterns, we calculate the accuracy of LogAnomaly without (w/o) template vector sequence which captures the sequential pattern of logs, as well as the accuracy of LogAnomaly without (w/o) count vector extracting the quantitative pattern of logs. As shown in Figure 6 and Figure 7, compared to LogAnomaly, LogAnomaly without (w/o)
Table 3: The detailed information of the BGL dataset for evaluating the performance of LogAnomaly in online anomaly detection

<table>
<thead>
<tr>
<th># template in training logs</th>
<th># template in detection logs</th>
<th># unmatched logs by training templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>257</td>
<td>503</td>
<td>299,174</td>
</tr>
</tbody>
</table>

Table 4: The accuracy of LogAnomaly and Deeplog in online anomaly detection

<table>
<thead>
<tr>
<th>Methods</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepLog</td>
<td>0.3817</td>
<td>0.9768</td>
<td>0.5489</td>
</tr>
<tr>
<td>LogAnomaly</td>
<td>0.8039</td>
<td>0.9319</td>
<td>0.8632</td>
</tr>
</tbody>
</table>

4.3 Evaluation of Online Anomaly Detection

As described in Section 3.2, template2Vec not only improves the accuracy of LogAnomaly by leveraging the semantic information of log templates, but also addresses the challenges of new types of log messages by matching new templates to existing templates. Usually, service systems often generate new types of logs and thus new log templates, due to frequent software or firmware updates aiming to introduce new features or fix bugs of previous versions. If an anomaly detection method cannot cope with new templates, it will generate false alarms, for the reason that the new templates can generate new patterns which match no historical normal pattern.

We evaluate the performance of LogAnomaly in addressing new types of logs at runtime as follows. As shown in Table 3, the front 50% (according to the timestamps of logs) of the BGL dataset is used as the training set, which includes 257 log templates, and the rest 50% involving 503 templates is used as the testing set. More specifically, 299,174 logs in the testing set cannot match any existing templates. Because both PCA and Invariant Mining are offline methods, and LogCluster performs badly in offline scenario as shown in Section 4.2, we compare LogAnomaly with Deeplog in this section. Note that we do not provide any manual feedback to Deeplog and LogAnomaly after they are trained.

Table 4 shows the accuracy of LogAnomaly and Deeplog in online anomaly detection. Although Deeplog achieves relatively high recall, its precision is much lower than that of LogAnomaly. More specifically, Deeplog generates 3.15 times of false alarms compared to LogAnomaly, which brings much unnecessary work to operators. With template2Vec, LogAnomaly is able to match a new type of log message to an existing template vector, and thus it is more accurate in online anomaly detection.

4.4 Case Study

To further evaluate the performance of LogAnomaly, we here show a single anomaly case that happened on an aggregation switch deployed in a top cloud service provider from 15:00, Oct 13 2018 to 1:16, Oct 14 2018. LogAnomaly, Invariants Mining, and Deeplog are deployed to detect anomalies based on logs for this switch during Sep 25 2018 to Oct 19, 2018. More specifically, as shown in Figure 8, the traffic forwarded by this switch dropped from 15:00, Oct 13 2018, and the services provided by this switch were impacted from 22:15, Oct 13 2018, until the switch recovered at 1:16, Oct 14 2018. LogAnomaly generated 29 alarms, all of which began from 15:59, Oct 13 2018, and ended before 1:16, Oct 14 2018. Consequently, LogAnomaly successfully detected this anomaly and generated no false alarm. DeepLog, however, generated 43 alarms during its deployment period (Sep 25 to Oct 19), 15 of which were false ones. Moreover, Invariant Mining generated five alarms, and three of them were false.

We can see that, in real-world cases, LogAnomaly achieved better accuracy than Invariants Mining and Deeplog, for the following reason. There are more than 10,000 log templates in this cloud service provider. A large number of log templates having different template indexes represent very similar events. Neither Invariants Mining nor Deeplog was able to leverage the similarities of log templates, whereas LogAnomaly successfully extracted these similarities by using template2Vec to learn templates’ semantic information.

5 Conclusion

Logs are one of the most valuable data sources for system operation. In this paper, we propose LogAnomaly, a unified log anomaly detection system that leverages a novel template2Vec method to extract the semantic information of log templates and address the challenges posed by new types of logs at runtime and combines the sequential and quantitative patterns of logs. Extensive experiments strongly demonstrate LogAnomaly’s superior performance. The real-world case study further shows that the novel template2Vec method successfully extracts the semantic information of logs.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. We also thank Ya Su, Yixuan Zhang, Yu Zhou, Yimin Zhou for their helpful suggestions and thank Juexing Liao for proofreading this paper. The work was supported by the BNRist, Okawa Research Grant, the Fundamental Research Funds for the Central Universities (Grant No. 63191427), National Natural Science Foundation of China (Grant Nos. 61772307 and 61402257), National Key R&D Program of China (Grant No. 2017YFB0801700), and CERNET Innovation Project (Grant No. NGII20180121).
References


