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

Conference Paper · August 2019

DOI: 10.24963/ijcai.2019/658

11 authors, including:

Weibin Meng
Tsinghua University
8 PUBLICATIONS **20** CITATIONS
SEE PROFILE

Ying Liu
Pharmaron
443 PUBLICATIONS **6,354** CITATIONS
SEE PROFILE

Shenglin Zhang
Nankai University
19 PUBLICATIONS **84** CITATIONS
SEE PROFILE

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

Weibin Meng[1,5], **Ying Liu**[1,5], **Yichen Zhu**[2], **Shenglin Zhang**[3*], **Dan Pei**[1,5], **Yuqing Liu**[3], **Yihao Chen**[1,5], **Ruizhi Zhang**[4], **Shimin Tao**[4], **Pei Sun**[4] and **Rong Zhou**[4]

[1]Tsinghua University
[2]University of Toronto
[3]Nankai University
[4]Huawei
[5]Beijing National Research Center for Information Science and Technology (BNRist)

mwb16@mails.tsinghua.edu.cn, liuying@cernet.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, joe.zhourong}@huawei.com

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 quantitive 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.

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.

L_1. 1537885119 IFNET/2/linkDown_active(l):CID=0x807a0405, alarmID=0x0852003; The interface status changes.
L_2. 1537885119 LACP/4/LACP_STATE_DOWN(l): CID=0x804804, PortName=40GE1/0/3; The LACP state is down. Reason = **The interface went down physically**.
L_3. 1537885130 DEVM/3/LocalFaultAlarm_clear(l): CID=0x852003, clearType= service_resume, The local fault alarm has resumed.
L_4. 1537885135 IFNET/2/linkDown_clear(l): CID=0x807a0405, alarmID=0x0852003; The interface status changes. Physical link is up, mainName=Eth-Trunk104.

L_5. 1539139152 IFNET/2/linkDown_active(l):CID=0x807a0406, alarmID=0x0852007; The interface status changes.
L_6. 1539138152 LACP/4/LACP_STATE_DOWN(l): CID=0x804807, PortName=40GE1/0/3; The LACP state is down. Reason = **No LCAPDUs were received**.
L_7. 1539138164 DEVM/3/LocalFaultAlarm_clear(l): CID=0x852004, clearType= service_resume, The local fault alarm has resumed.
L_8. 1539138164 IFNET/2/linkDown_clear(l): CID=0x807a0406, alarmID=0x0852007; The interface status changes. Physical link is up, mainName=Eth-Trunk104.

Figure 1: Switch logs of two normal link flappings

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 L_1 and L_2, 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 L_4. 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 mechods 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_1 to L_4. When a system generates L_6 (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.

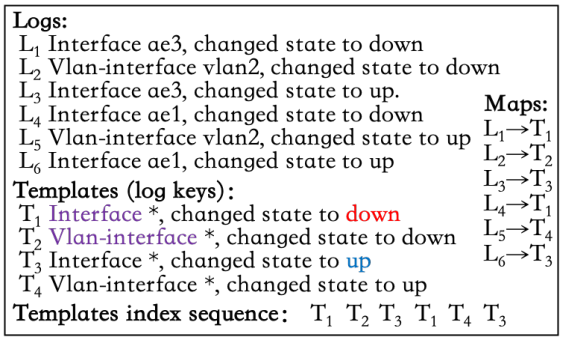


Figure 2: Logs and templates

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.,* "$ae1$" and "$ae3$", 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, achieveing 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.*, $count(T_1) = count(T_3)$ for Figure 2) hold true within the window.

[Zhang *et al.*, 2016] and DeepLog [Du *et al.*, 2017] are

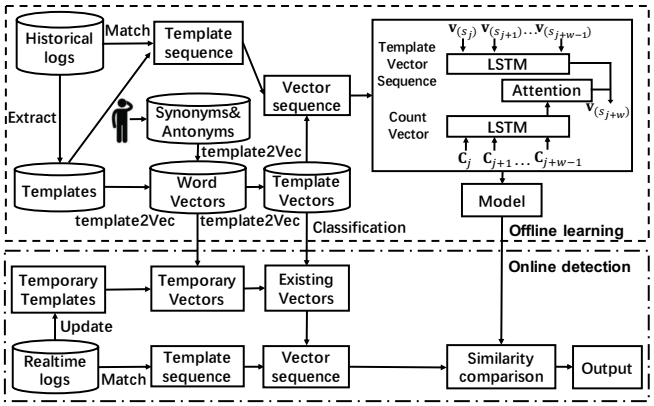


Figure 3: The framework of LogAnomaly

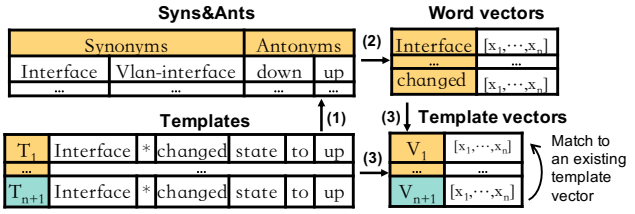


Figure 4: Examples of template2Vec.

designed to detect sequential anomalies in an unsupervised manner. They utilize LSTM to predict the next logs likely to appear after seeing logs, and an anomaly is declared if the actual log is unexpected according to the prediction.

Because anomaly labels are often unavailable in practice, thus unsupervised methods are much more practical in real-world service systems. As mentioned in Section 1, existing unsupervised approaches suffer from some common drawbacks. First, they detect anomalies based on the log template *indexes* (*i.e.*, using 3 to represent T_3 "$Interface\ *, changed\ state\ to\ up$"), which does not utilize the semantic information in the logs. Second, they are designed to capture *either* the sequential *or* quantitative patterns of logs, but not *both*.

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 some patterns. Therefore, logs, which are generated by these programs, have some patterns in types, temporal orders, event count frequencies, and quantitative relationship between counts of different events. Our core idea is then to use deep learning to offline learn both sequential and quantitative patterns of the logs, each of which is represented by its template's semantics (as opposed to just the index of the template); in online detection, the realtime logs which violated the learned patterns are considered anomalous.

The design of our proposed LogAnomaly is shown in Figure 3. In the offline learning component, LogAnomaly first leverages FT-Tree (see Section 2 for more details) 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.

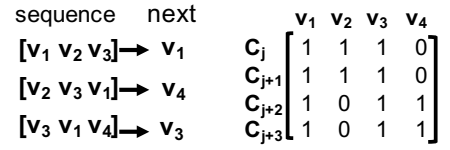
3.2 Template2Vec

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 T_1 and the word "up" in template T_3 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

Relations	$Word_1$	$Word_2$	Add Methods
Synonyms	down interface	low port	WordNet Operators
Antonyms	down powerDown	up powerOn	WordNet Operators

Table 1: Examples of synonyms and antonyms in logs



(a) Sequential pattern (b) Quantitative pattern

Figure 5: The sequential and quantitative patterns of the templates in Figure 2

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 $\Omega = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\}$ 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 $\mathbf{S} = (s_1, s_2, ..., s_m)$, suppose that $\mathbf{S}_j = (s_j, s_{j+1}, ..., s_{j+w-1})$ 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 $[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ is likely to be $\mathbf{v}_1$. 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 $\mathbf{S}_j$ is $\mathbf{V}_j = (\mathbf{v}_{(s_j)}, \mathbf{v}_{(s_{j+1})}, ..., \mathbf{v}_{(s_{j+w-1})})$, where $\mathbf{v}_{(s_i)}$ is the template vector of s_i, and $\mathbf{v}_{(s_i)} \in \Omega$. Therefore, $\mathbf{V}_j$ is input into the LSTM model for $\mathbf{S}_j$ 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 $s_i \in \mathbf{S}_j$, we calculate the count vector of the log sequence $(s_{i-w+1}, s_{i-w+2}, ..., s_i)$, which is denoted as $\mathbf{C}_i = (c_i(\mathbf{v}_1), c_i(\mathbf{v}_2), ..., c_i(\mathbf{v}_n))$, where $c_i(\mathbf{v}_k)$ is the number of $\mathbf{v}_k$ in the template vector sequence $(\mathbf{v}_{(s_{i-w+1})}, \mathbf{v}_{(s_{i-w+2})}, ..., \mathbf{v}_{(s_i)})$, and $\mathbf{v}_k \in \mathbf{V}$. Finally, $\mathbf{C}_j, \mathbf{C}_{j+1}, ..., \mathbf{C}_{j+w-1}$ is input into LSTM to learn the quantitative pattern of $\mathbf{S}_j$.

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
HDFS	38.7 hours	11,175,629	16,838(blocks)
BGL	7 months	4,747,963	348,460(logs)

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 [Du *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. Precision $= \frac{\text{TP}}{\text{TP+FP}}$, Recall $= \frac{\text{TP}}{\text{TP+FN}}$, F1 score $= \frac{\text{2*Precision*Recall}}{\text{Precision + Recall}}$.

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

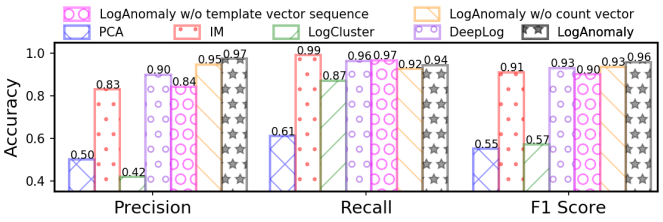


Figure 6: Accuracy on the BGL dataset

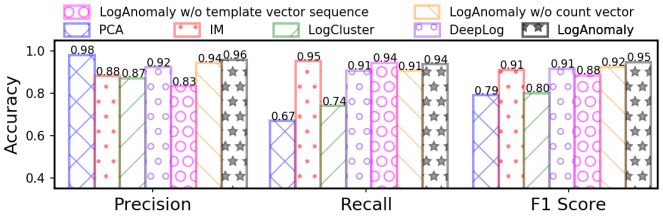


Figure 7: Accuracy on the HDFS dataset

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 achieve high recall on both the BGL dataset and the HDFS dataset. For example, 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.,* T_1 and T_2 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 fall.

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)

# template in training logs	# template in detection logs	# unmatched logs by training templates
257	503	299,174

Table 3: The detailed information of the BGL dataset for evaluating the performance of LogAnomaly in online anomaly detection

Methods	Precision	Recall	F1 score
DeepLog	0.3817	0.9768	0.5489
LogAnomaly	0.8039	0.9319	0.8632

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

template vector sequence has much low precisions on both the BGL dataset (0.84) and the HDFS dataset (0.83) compared to LogAnomaly (0.97 and 0.96 on the two datasets, respectively). Therefore, the sequential pattern of LogAnomaly prevents false alarms from disturbing operators. Similarly, the quantitative pattern, which are represented with count vectors, improves both precision and recall for LogAnomaly on the two datasets.

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.

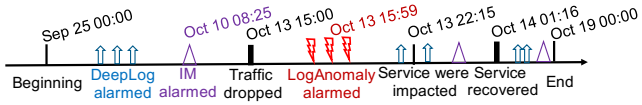


Figure 8: Timeline of switch anomaly case study

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

[Bu *et al.*, 2018] Jiahao Bu, Ying Liu, Shenglin Zhang, Weibin Meng, Qitong Liu, Xiaotian Zhu, and Dan Pei. Rapid deployment of anomaly detection models for large number of emerging kpi streams. In *IEEE IPCCC*, 2018.

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

[He *et al.*, 2016] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218. IEEE, 2016.

[He *et al.*, 2018] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. Identifying impactful service system problems via log analysis. In *ESEC/FSE*, pages 60–70. ACM, 2018.

[Khatuya *et al.*, 2018] Subhendu Khatuya, Niloy Ganguly, Jayanta Basak, Madhumita Bharde, and Bivas Mitra. Adele: Anomaly detection from event log empiricism. *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 2114–2122, 2018.

[Liang *et al.*, 2007] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. Failure prediction in ibm bluegene/l event logs. In *Data Mining, ICDM 2007*, pages 583–588. IEEE, 2007.

[Lin *et al.*, 2016] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 102–111. ACM, 2016.

[Lou *et al.*, 2010] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. Mining invariants from console logs for system problem detection. In *USENIX Annual Technical Conference*, pages 23–25, 2010.

[Ma *et al.*, 2018] Minghua Ma, Shenglin Zhang, Dan Pei, Xin Huang, and Hongwei Dai. Robust and rapid adaption for concept drift in software system anomaly detection. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 13–24. IEEE, 2018.

[Meng *et al.*, 2018] Weibin Meng, Ying Liu, Shenglin Zhang, Dan Pei, Hui Dong, Lei Song, and Xulong Luo. Device-agnostic log anomaly classification with partial labels. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–6. IEEE, 2018.

[Messaoudi *et al.*, 2018] Salma Messaoudi, Annibale Panichella, and Domenico Bianculli. A search-based approach for accurate identification of log message formats. In *Proceedings of the 26th IEEE/ACM International Conference on Program Comprehension*. ACM, 2018.

[Mikolov *et al.*, 2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[Miller, 1995] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[Nandi *et al.*, 2016] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B Dasgupta, and Subhrajit Bhattacharya. Anomaly detection using program control flow graph mining from execution logs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 215–224. ACM, 2016.

[Nguyen *et al.*, 2016] Kim Anh Nguyen, Sabine Schulte im Walde, and Ngoc Thang Vu. Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction. *arXiv preprint arXiv:1605.07766*, 2016.

[Oliner and Stearley, 2007] Adam J. Oliner and Jon Stearley. What supercomputers say: A study of five system logs. *IEEE International Conference on Dependable Systems and Networks (DSN'07)*, pages 575–584, 2007.

[Satpathi *et al.*, 2018] Siddhartha Satpathi, Supratim Deb, R Srikant, and He Yan. Learning latent events from network message logs: A decomposition based approach. *arXiv preprint arXiv:1804.03346*, 2018.

[Xu *et al.*, 2009] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Largescale system problem detection by mining console logs. *Proceedings of SOSP'09*, 2009.

[Zhang *et al.*, 2015] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, and Zhi Zang. Rapid and robust impact assessment of software changes in large internet-based services. In *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 1–13, 2015.

[Zhang *et al.*, 2016] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. Automated it system failure prediction: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1291–1300. IEEE, 2016.

[Zhang *et al.*, 2017] Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping Qu, et al. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2017.

[Zhang *et al.*, 2018a] Shenglin Zhang, Ying Liu, Weibin Meng, Zhiling Luo, Jiahao Bu, Sen Yang, Peixian Liang, Dan Pei, Jun Xu, Yuzhi Zhang, et al. Prefix: Switch failure prediction in datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(1):2, 2018.

[Zhang *et al.*, 2018b] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, Zhi Zang, Xiaowei Jing, and Mei Feng. Funnel: Assessing software changes in web-based services. *IEEE Transactions on Services Computing*, 11(1):34–48, 2018.