Feedback-Aware Anomaly Detection Through Logs for Large-Scale Software Systems

Abstract: One particular challenge for large-scale software systems is anomaly detection. System logs are a straightforward and common source of information for anomaly detection. Existing log-based anomaly detectors are unusable in real-world industrial systems due to high false-positive rates. In this paper, we incorporate human feedback to adjust the detection model structure to reduce false positives. We apply our approach to two industrial large-scale systems. Results have shown that our approach performs much better than state-of-the-art works with 50% higher accuracy. Besides, human feedback can reduce more than 70% of false positives and greatly improve detection precision.

Keywords: human feedback; log-based anomaly detection; system log

HAN Jing[1], JIA Tong[2], WU Yifan[2], HOU Chuanjia[2], LI Ying[2]

(1. ZTE Corporation, Shenzhen 518057, China;
2. Peking University, Beijing 100091, China)

DOI: 10.12142/ZTECOM.202103011

http://kns.cnki.net/kcms/detail/34.1294.TN.20210730.1102.001.html, published online July 30, 2021

Manuscript received: 2021-02-04

Citation (IEEE Format): J. Han, T. Jia, Y. F. Wu, et al., "Feedback-aware anomaly detection through logs for large-scale software systems," *ZTE Communications*, vol. 19, no. 3, pp. 88 – 94, Sept. 2021. doi: 10.12142/ZTECOM.202103011.

1 Introduction

Large-scale software systems face one particular challenge which is anomaly detection. System logs provide a straightforward and common information source for anomaly detection. Typically, administrators manually check log files and search for problem-related log entries, which is error-prone and time-tedious. To reduce human efforts, researchers have proposed many automatic log-based anomaly detectors[1-19]. However, these detectors are ineffective in real-world industrial systems. First, most detectors typically operate by identifying statistical outliers. The utility of a particular detector for a system depends on how well its statistical outliers align with system anomaly symptoms. In general, the gap between statistical outliers and real system anomalies can result in high false-positive rates and easily render an anomaly detector unusable. Second, new types of anomalies may arise during system updates and conflict with existing anomaly detectors to produce false positives. Third, heterogeneous and complex log data contains massive noise. This noise may mislead detectors and further increase false positives.

One way to reduce the false-positive rate is to build domain knowledge into a detector. For example, a designer might apply domain expertise to label training logs that are more likely to produce correct anomalies and/or filter anomalies based on semantically defined white lists. Unfortunately, this requires significant expertise in both the system and anomaly detection. Besides, a large number of logs from industrial large-scale systems are almsot impossble to label; e.g., a Microsoft online service system even generates over one petabyte (PB) of logs every day[20].

In this paper, we consider an approach to reduce false positives based on incorporating human feedback. In our settings of feedback-aware anomaly detection, humans only provide feedback about whether the detected anomaly is false positive or not. This feedback is used by the detector to adjust the anomaly detection model structure. This approach has the advantage of an easy and concise feedback process with little overhead on time. The main contributions of this paper includes:

1) To the best of our knowledge, we are the first to incorporate human feedback to reduce false positives for the log-based anomaly detection task.

2) We propose a feedback-aware online anomaly detection approach that builds a graph model from an online log stream and adjusts the graph structure through human feedback.

3) We apply our approach to two industrial large-scale systems. Results have shown that human feedback can reduce

This work was supported by ZTE Industry-University-Institute Cooperation Funds under Grant No. 20200492.

most false positives and greatly improve detection precision.

The rest paper is organized as follows. Section 2 discusses the related work, Section 3 proposes the approach, Section 4 shows the experiment results, and Section 5 concludes this paper.

2 Related Work

2.1 Human-in-the-Loop Anomaly Detection

In existing works, incorporating human feedback into anomaly detection has been introduced. These works leverage the idea of active learning and focus on tuning the weights and scores in machine learning models. For instance, the online mirror descent (OMD) algorithm[21] associates a convex loss function to each feedback response which rewards the anomaly score. Active anomaly discovery (AAD) algorithm[22-23] defines and solves an optimization problem based on all prior feedback, which results in new weights for the model.

2.2 Log-Based Anomaly Detection

Log-based anomaly detection first parses logs into log templates based on static code analysis or clustering mechanism, and then builds anomaly detection models. These models include template frequency-based model, graph-based model, and deep learning-based model. The template frequency-based model[1-4] usually counts the number of different templates in a time window and sets up a vector for each time window. Then it utilizes methods such as machine learning algorithms to distinguish outliers.

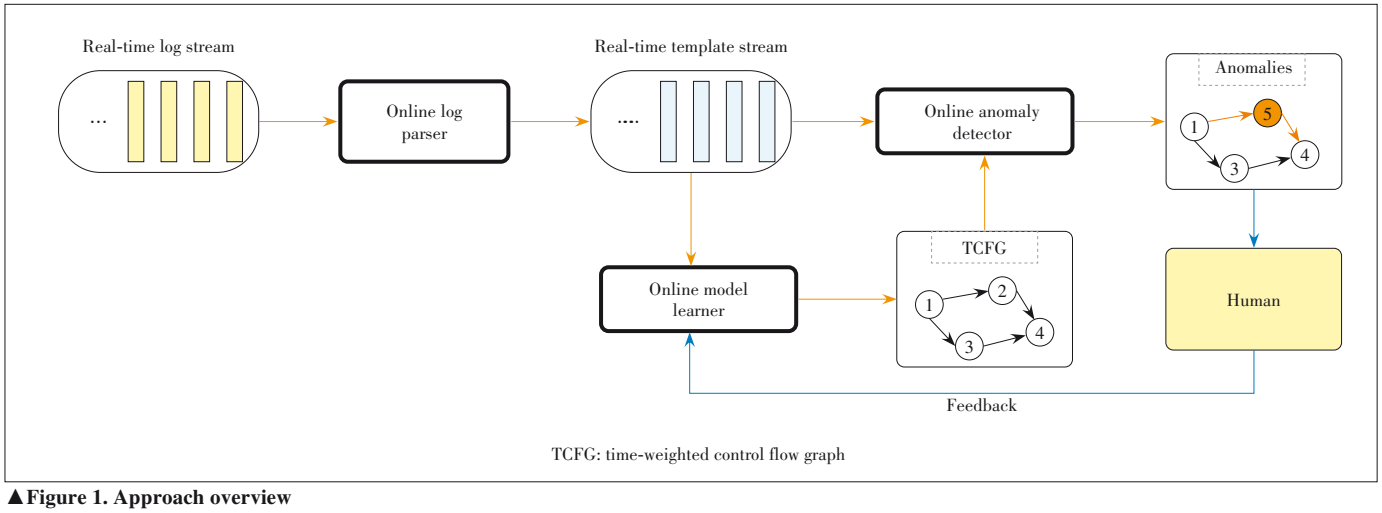
This model sacrifices the abundant information and the diagnosis ability of logs and it is not accurate and efficient. Thus it cannot provide help for problem identification and diagnosis. The graph-based model[5-17] is the current research hotspot. It extracts template sequence at first and then generates a graph-based model to compare with log sequences in the production environment to detect conflicts. This model has three advantages. First, it can diagnose problems deeply buried in log sequences, for example, performance degradation. Second, it can provide engineers with the context log messages of problems. Third, it can provide engineers with the correct log sequence and tell engineers what should have happened. The deep learning-based model[18-19] leverages long short-term memory (LSTM) to model the sequence of templates. This model takes a long time for training and inference, and thus cannot support online anomaly detection and diagnosis.

3 Approach

3.1 Overview

To solve the problems mentioned above, we design a human feedback-aware anomaly detection approach, called LogFlash, as shown in Fig. 1. The input is an online log stream $\boldsymbol{l} := (l_1, l_2, l_3, \ldots)$, which is a log entry. Our approach consists of three main components, namely the online log parser, the online model learner, and the online anomaly detector. In the online log parser, multiple log templates are mined from the log stream and each log entry is replaced by its corresponding template. In this way, the log stream is transformed into a template stream $\boldsymbol{p} := (p_1, p_2, p_3, \ldots)$. This template stream then goes through online model learner and online anomaly detector concurrently. The online model learner infers and updates a graph model called time-weighted control flow graph (TCFG) through mining the template stream. The online anomaly detector utilizes the latest TCFG model to detect anomalies in the template stream. Humans provide false positives in anomalies as feedback to the online model learner. The learner then adjusts the TCFG structure based on the feedback.

We leverage the existing online template mining algorithm[24] in the online log parser. Due to space limitations, we will only describe the TCFG model, online model learner, online anomaly detector, and human feedback loop.



▲Figure 1. Approach overview

3.2 Time-Weighted Control Flow Graph

A TCFG is a directed graph consisting of edges and nodes and each edge has a time weight recording the transition time. The TCFG model stitches together various log templates and represents the healthy state of the baseline system. It is used to flag deviations from expected behaviors at runtime. A template is an abstraction of a print statement in a source code, which manifests itself in logs with different embedded parameter values in different executions. Represented as a set of invariant keywords and parameters (denoted by parameter placeholder *), a template can be used for summarization of multiple log lines. The TCFG is such a graph where the nodes are templates and the edges represent the transition from one template to another. Besides, every log has a timestamp indicating its print time, and thus the difference between two log timestamps represents the program execution time between the two logs. The time weight on each edge in the TCFG records the longest normal transition time between two templates. If the execution time between two logs exceeds the time weight, it means the system is suffering from performance problems.

Fig. 2 shows an example of log templates and TCFG model. Each log has some invariant keywords and some variable parameters (shown in green), and log templates only reserve invariant keywords. Nodes in the TCFG are different log templates. Edges represent how each request flow passes between nodes, and the weight of edges indicates the transition time between two nodes.

3.3 Online Model Learner

We aim to construct a TCFG model in a black-box manner with only the template stream p. Our key idea is to define a dynamic pairwise transition rate $\alpha_{j,i}$ which models how frequently a request flows from template j to template i and trains/updates the transition rate $\alpha_{j,i}$ overtime with template stream p.

We further define $f\left(t_i|t_j,\alpha_{j,i}\right)$ to be the conditional likelihood of transition between template j and template i, where t_j and t_i are the timestamps of two occurrences of template j and template i in p. We assume the conditional likelihood depends on the transition time $\left(t_j,t_i\right)$ and the transition rate $\alpha_{j,i}$. To model this parametric likelihood, we first conduct a statistical analysis of the distribution of template transitions.

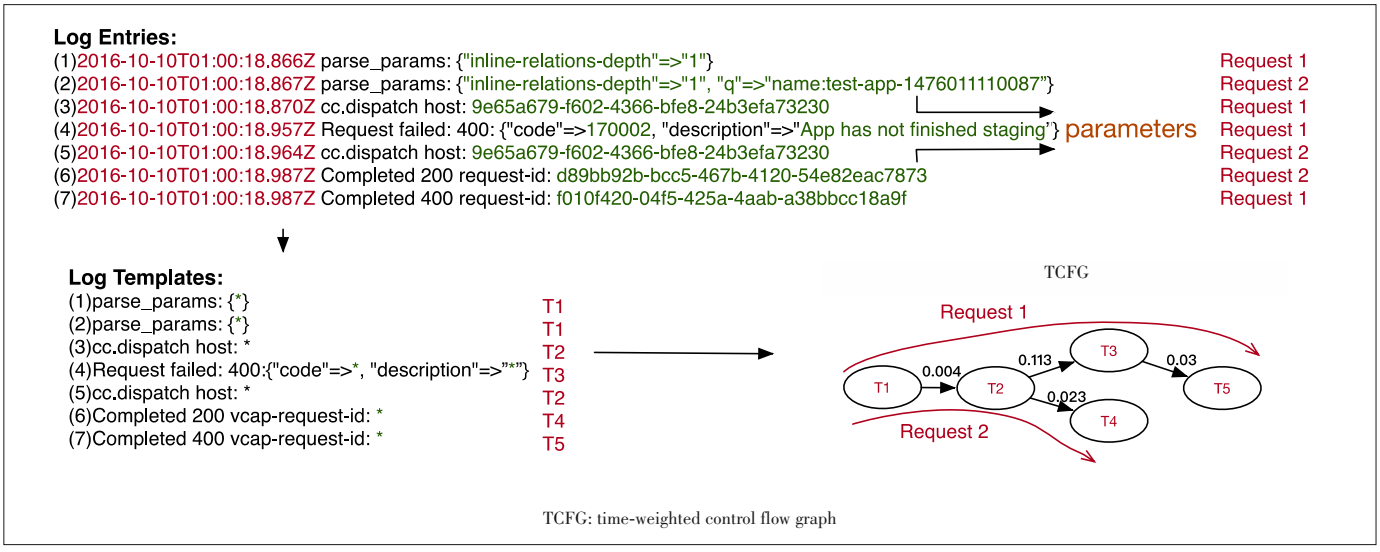
We collect system logs of 5 minutes from an industrial cloud system Ada. Then we record the transition time between every occurrence of two neighboring templates in the same request by calculating the difference of their timestamps. Next, we count the number of occurrences with the same transition time and plot the distribution of each template transition. Results are shown in Fig. 3. The distributions of these transitions show obvious long-tail distribution characteristics and the most transitions cost less than 0.2 norm-value of time.

Based on the above observations, the power-law likelihood is appropriate to model $f\left(t_i|t_j,\alpha_{j,i}\right)$, that is:

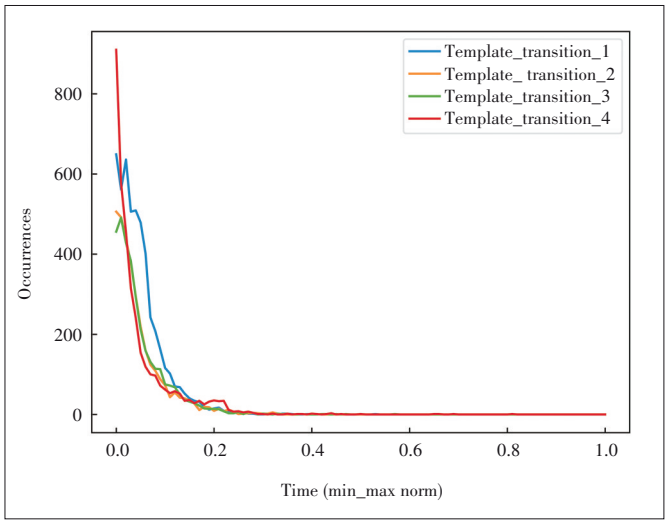
$$f\left(t_i|t_j,\alpha_{j,i}\right)=\begin{cases}\dfrac{\alpha_{j,i}}{\delta}\left(\dfrac{t_i-t_j}{\delta}\right)^{-1-\alpha_{j,i}} & \text{if } t_j+\delta<t_i\\ 0 & \text{otherwise}\end{cases}, \tag{1}$$

where δ states the minimum transition time from template j to template i. In Section 4, the power-law distribution proves to be generic enough to adapt anomaly detection methods to testing logs from diverse industrial systems. Then we apply network inference algorithm to train the structure of TCFG.

1) Template stream likelihood. In the template stream p, transitions from different templates to a certain template are independent, that is, each occurrence of template i can only be transmitted from the occurrence of one parent template. Then the likelihood of occurrence of template i at time t_i, giv-



▲Figure 2. Log templates and TCFG model



▲ **Figure 3. Template transition distributions of an industrial software system Ada**

en a collection of previous occurred templates $(t_1,...,t_N|t_k \leqslant t_i)$, results from summing over the likelihood of the mutually disjoint transition from each previously occurred template to template i:

$$f\left(t_i|t_1,...,t_{N_{t_i}},A\right)=\sum_{j:t_j<t_i} f\left(t_i|t_j,\alpha_{j,i}\right)\times\prod_{k:k\neq j,t_k<t_i} S\left(t_i|t_k,\alpha_{k,i}\right), \quad (2)$$

where $A=\{\alpha_{j,i}|i,j=1,...,N,i\neq j\}$, and $S\left(t_i|t_k,\alpha_{k,i}\right)$ is a defined survival function of transition $j\rightarrow i$ as

$$S\left(t_i|t_k,\alpha_{k,i}\right)=1-F\left(t_i|t_k,\alpha_{k,i}\right), \quad (3)$$

where $F\left(t_i|t_k,\alpha_{k,i}\right)=\int_{t_j}^{t_i} f\left(t|t_j,\alpha_{j,i}\right)dt$ is the cumulative transition density function computed from the transition likelihood.

To simplify the modeling process, we assume that transitions are conditionally independent, given a set of parent templates. The likelihood of all transitions in the template stream is

$$f\left(\boldsymbol{t}^{\leqslant T},A\right)=\prod_{t_i\leqslant T} f\left(t_i|t_1,...,t_{N_{t_i}},A\right), \quad (4)$$

where $\boldsymbol{t}^{\leqslant T}$ denotes that the time of template stream is up to T. After plugging Eq. (2) into Eq. (4) and removing the condition $k\neq j$, the product result is independent of j:

$$f\left(\boldsymbol{t}^{\leqslant T},A\right)=\prod_{i:t_i\leqslant T}\prod_{k:t_k<t_i} S\left(t_i|t_k,\alpha_{k,i}\right)\times\sum_{j:t_j<t_i}\frac{f\left(t_i|t_j,\alpha_{j,i}\right)}{S\left(t_i|t_j,\alpha_{j,i}\right)}. \quad (5)$$

The fact that some templates are not shown in the observation window is also informative. We therefore add multiplicative survival terms to Eq. (5) and rearrange it with hazard function[25] or instantaneous transition rate of transition $j\rightarrow i$ as $H\left(t_i|t_j,\alpha_{j,i}\right)=f\left(t_i|t_j,\alpha_{j,i}\right)/S\left(t_i|t_j,\alpha_{j,i}\right)$. Then the likelihood of the template stream is reformulated as

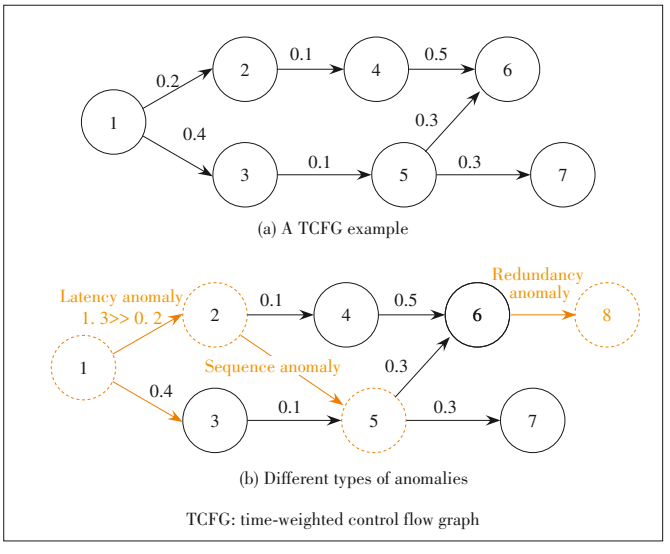
$$f\left(\boldsymbol{t},A\right)=\prod_{i:t_i\leqslant T}\prod_{m:t_m>T} S\left(T|t_i,\alpha_{i,m}\right)\times\prod_{k:t_k<t_i} S\left(t_i|t_k,\alpha_{k,i}\right)\left(\sum_{j:t_j<t_i} H\left(t_i|t_j,\alpha_{j,i}\right)\right). \quad (6)$$

2) TCFG structure inference problem. Our purpose is to infer a TCFG structure that is most possible to generate the template stream p. Given a TCFG with constant edge transition rate A, the TCFG structure inference problem problem reduces to solving a maximum likelihood problem:

$$\begin{aligned}&\text{maximize}_A \quad \log f\left(t,A\right)\\&\text{subject to} \quad \alpha_{j,i}\geqslant 0, i,j=1,...,N,i\neq j\end{aligned}, \quad (7)$$

where $A=\{\alpha_{j,i}|i,j=1,...,N,i\neq j\}$ are the edge transitions we aim to train. The edges in TCFG are those pairs of templates with transition rates $\alpha_{j,i}\geqslant 0$.

To support online model update, we generalize the inference problem to dynamic TCFG structure with edge transition rates $A(t)$ that may change over time. To this aim, we first split the template stream p to a set of sub-streams $\boldsymbol{c}=\left(c_1,c_2,c_3,...\right)$ based on the arrival of new templates. Given a time window size w, each time a template i arrives, we split out a sub-stream in which i is the latest template. An example is shown in Fig. 4. At time t_1, log stream in the red block is the current sub-stream. At time t_2, a new template T_2 is observed and the current sub-stream becomes $\{T_3,T_4,T_3,T_2\}$. When it comes to time t_3 when T_5 is observed, the current sub-stream becomes $\{T_4,T_3,T_2,T_5\}$. In this way, at any given



▲ **Figure 4. A TCFG example and different types of anomalies**

time t, we solve the maximum likelihood problem over the set of sub-streams:

$$\begin{aligned} &\text{maximize}_{A(t)} \quad \sum_{c \in \boldsymbol{c}} f(t^c, A(t)) \\ &\text{subject to} \quad \alpha_{j,i}(t) \geqslant 0, i,j = 1,...,N, i \neq j, \end{aligned} \tag{8}$$

where $c \in \boldsymbol{c}$. Next, we show how to efficiently solve the above optimization problem for all time points t.

3) Training method. The problem defined by Eq. (8) is serious for the power-law transition model. Therefore, we aim to find optimal training solution at any given time t. Since in the condition of power-law model, the edge transition rates usually vary smoothly, classical stochastic gradient descent[26] can be a perfect method for our training as we can use the inferred TCFG structure from the previous time step as initialization for the inference procedure in the current time step. The training phase uses iterations of the form:

$$\alpha_{j,i}^k(t) = \left(\alpha_{j,i}^{k-1}(t) - \gamma \nabla_{\alpha_{j,i}} L_c\left(A^{k-1}(t)\right)\right)^+, \tag{9}$$

where k is the iteration number, $\nabla_{\alpha_{j,i}} L_c(\cdot)$ is the gradient of the log-likelihood $L_c(\cdot)$ of sub-stream c with respect to the edge transition rate $\alpha_{j,i}$, γ is the update step size, and $(z)^+ = \max(0,z)$. The computations of log survival function, hazard function and gradient of sub-stream c for power-law model in Eq. (1) are given in Table 1.

Importantly, in each iteration of the training phase, we only need to compute the gradients $\nabla_{\alpha_{j,i}} L_c\left(A^k\right)$ for edges between template j and template i, as node j has been observed in sub-stream $\boldsymbol{c}$, and the iteration cost and convergence rate are independent of $|c|$.

3.4 Online Anomaly Detector

The basic idea for anomaly detection is to compare the log stream with TCFG to find the deviation. We first define three types of deviations/anomalies, namely sequence anomaly, redundancy anomaly, and latency anomaly. A sequence anomaly is raised when the log that follows the occurrence of a parent node cannot be mapped to any of its children. A redundancy anomaly is raised when unexpected logs that cannot be mapped to any node in the TCFG occur. A latency anomaly is raised when the child of a parent node is seen but the transition time exceeds the time weight recorded on the edge. Fig. 4 shows an example of different types of anomalies. Fig. 4(a) is an example of TCFG with 7 nodes. As shown in Fig. 4(b), suppose the transition time between Node 1 and Node 2 exceeds the time weight 0.2, they suffer from a latency anomaly. Node 5 appears after Node 2 unexpectedly and suffers from a sequence anomaly. Node 8 appears after Node 6 while Node 8 is a new template that has not been recorded in the TCFG, and thus a redundancy anomaly occurs.

▼**Table 1. Computations of transition likelihood for power-law model**

Computation Entity	Computation Method
Log survival function: $\log S\left(t_i \mid t_k, \alpha_{k,i}\right)$	$-\alpha_{j,i} \log\left(\frac{t_i - t_j}{\delta}\right)$
Hazard function: $H\left(t_i \mid t_j, \alpha_{j,i}\right)$	$\alpha_{j,i} \cdot \frac{1}{t_i - t_j}$
Gradient for unobserved ones in $\boldsymbol{c}$: $\nabla_{\alpha_{j,i}} L_c(A)$	$\log\left(\frac{T - t_j^c}{\delta}\right)$
Gradient for observed ones in $\boldsymbol{c}$: $\nabla_{\alpha_{j,i}} L_c(A)$	$\log\left(\frac{t_i^c - t_j^c}{\delta}\right) - \frac{(t_i^c - t_j^c)^{-1}}{\sum_{k: t_k^c < t_i^c} \alpha_{k,i} (t_i^c - t_k^c)^{-1}}$

3.5 Human Feedback Handling

As mentioned before, users report false positives in anomalies through detection results webpage as human feedback. The online model learner receives the feedback and adjust the TCFG based on the feedback. For different types of anomalies, the online model learner takes different operations. These operations are shown in Algorithm 1.

Algorithm 1. Human Feedback Handling Algorithm

Input: Human feedback **Anomaly**.
Definition: TCFG denotes the current TCFG model
1. **if Anomaly**. type = "Sequence"
2. **then TCFG**. addEdge (**Anomaly**. parentNode, **Anomaly**. childNode)
3. **if Anomaly**. type = "Redundancy"
4. **then TCFG**. addNode (**Anomaly**. redundantNode)
5. **if Anomaly**. type = "Latency"
6. **then TCFG**. setTimeWeight (**Anomaly**. parentNode, **Anomaly**. childNode, **Anomaly**. transitionTime)

• Sequence anomaly. A sequence anomaly is raised when the log that follows the occurrence of a parent node cannot be mapped to any of its children. If a sequence anomaly is a false positive, it means the log that follows the occurrence of the parent node should be its child. In other words, the transition between the parent and the child has not been correctly learned. For instance, in Fig. 4(b) Node 5 appears after Node 2 unexpectedly and suffers from a sequence anomaly. If it is a false positive, it means there should be a transition edge from Node 2 to Node 5. Therefore, the online model learner takes the operation to add a transition edge from the parent and the child in TCFG.

• Redundancy anomaly. A redundancy anomaly is raised when unexpected logs occur that cannot be mapped to any node in TCFG. If a redundancy anomaly is a false positive, it means the template of the unexpected log should be in TCFG. For instance, in Fig. 4(b) Node 8 appears unexpectedly and raises a redundancy anomaly. If it is a false positive, it means Node 8 should be in the path. Therefore, online model learners take the operation to add the template of the unexpected log to TCFG.

• Latency anomaly. The time weight on each edge in the

TCFG records the longest normal transition time between two nodes. An intuitive way to determine time weight is to update the time weight once it meets a longer transition time in the log stream. However, it is hard to determine whether the longer transition time is a real latency anomaly or normal latency fluctuation. Therefore, we rely on human feedback to update the time weight. If a latency anomaly is a false positive, it means the time weight is too small and should be updated. Therefore the online model learner updates the time weight once it receives feedback of latency anomaly. For instance, in Fig. 4(b) the transition time between Node 1 and Node 2 exceeds the time weight 0.2, and then they suffer a latency anomaly. If it is a false positive, it means the transition time 1.3 is normal. Therefore, the online model learner takes the operation to update the time weight from 0.2 to 1.3.

4 Experiment and Evaluation

4.1 Experiment Setup

We test our approach on three industrial large-scale systems called Ada, Bob, and Dockerd. Ada is an online image identification and analytics system that serves thousands of users. Bob is a software distribution system for 5G stations and chipboards. It distributes upgrade or bug fixing patches to thousands of 5G chipboards. We collect system logs of two days from Ada and Bob and use logs for training. Dockerd is a component of a Platform as a Service (PaaS) platform which contains 10 components and 957 nodes producing more than 8.1 million system logs per day. We collect logs of 20 days with a size of 52.94 G to verify the effectiveness of our approach.

We choose the state-of-the-art log-based anomaly detection DeepLog[18] and LogSed[6] as baselines. DeepLog leverages LSTM to model template sequences and detect anomalies through computing the distance between observed templates and predicted templates. LogSed first proposes TCFG model and infers the TCFG model based on frequent sequence mining.

We use typical Recall and Precision as our evaluation metrics, which are defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{10}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{11}$$

where TP, FP, TN, FN are referred to as true positive, false positive, true negative, and false negative.

4.2 Evaluation Results

Logs in real-world industrial systems are much more complex and heterogeneous than lab systems, and it is very hard for today's anomaly detectors to produce satisfying results. Table 2 shows the evaluation results of Ada. Both LogSed and DeepLog show poor precision which means they produce lots of false positives. Our approach outputs 74 anomalies in which 31 anomalies are true positives leading to a precision of 0.42 without human feedback (statistics in parentheses). After we incorporate human feedback, our approach produces 36 anomalies. We also record the times of human feedback during training. Experts labeled 28 false positives as feedback to guide the system and the labeling task only costs about 5 minutes.

Bob is even more complex than Ada. Each 5G station and chipboard are in different environments with different network status, load status, etc. Logs of many processes such as network test, heartbeat, software download, reconnect, software security and consistency check are interleaved together. It is almost impossible for existing detectors to learn a usable model from such noisy system logs. As shown in Table 3, LogSed and DeepLog show a precision of 0.04 and 0.09 separately. Our approach detects 137 anomalies without human feedback in which 10 anomalies are true positives. After incorporating human feedback, the number of detected anomalies reduces to 13 leading to 0.77 precision. During training, experts label 52 false positives as feedback in total that costs about 15 minutes.

As evaluating the performance of the framework on Dockerd logs, our approach detects 1 515 sequence anomalies without human feedback, of which less than 160 are true positives. After dropping duplicates and incorporating human feedback, the accuracy rate increases to 0.82. In summary, our approach achieves much better precision than baseline works. Incorporating human feedback effectively reduces false positives and significantly improves model performance. Besides, the feedback process is very easy for experts and saves a lot of time.

5 Conclusions and Future Work

In this paper, we propose a feedback-aware anomaly detection approach. It builds a TCFG model to describe normal system status and incorporates human feedback to adjust the graph structure to reduce false positives. Experiment results on two industrial large-scale systems show that our approach enjoys much better precision than baselines. Besides, human feedback can significantly reduce false positives and improve model performance.

▼Table 2. Evaluation results of Ada

Approaches	Precision/%	Recall/%	#Human Feedback
LogSed[18]	0.34	1.00	/
DeepLog[6]	0.45	1.00	/
Our approach	0.86(0.42)	1.00	28

▼Table 3. Evaluation results of Bob

Approaches	Precision/%	Recall/%	#Human Feedback
LogSed[18]	0.04	0.89	/
DeepLog[6]	0.09	0.99	/
Our approach	0.77(0.07)	0.96	52

In the future, we will improve the human feedback handling process and perform more sophisticated tuning on the model with human feedback.

References

[1] LOU J G, FU Q, YANG S Q, et al. Mining invariants from console logs for system problem detection [C]//USENIX Annual Technical Conference. Berkeley, USA: USENIX, 2010
[2] OLINER A J, AIKEN A. Online detection of multi-component interactions in production systems [C]//2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN). Hong Kong, China: IEEE, 2011: 49 – 60. DOI: 10.1109/DSN.2011.5958206
[3] CHEN C, SINGH N, YAJNIK S. Log analytics for dependable enterprise telephony [C]//2012 Ninth European Dependable Computing Conference. Sibiu, Romania: IEEE, 2012: 94 – 101. DOI: 10.1109/EDCC.2012.14
[4] DU S Z, JIAN C. Behavioral anomaly detection approach based on log monitoring [C]//2015 International Conference on Behavioral, Economic and Socio-cultural Computing (BESC). Nanjing, China: IEEE, 2015: 188 – 194. DOI:10.1109/BESC.2015.7365981
[5] NANDI A, MANDAL A, ATREJA S, et al. Anomaly detection using program control flow graph mining from execution logs [C]//The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco, USA: ACM, 2016: 215 – 224. DOI:10.1145/2939672.2939712
[6] JIA T, YANG L, CHEN P F, et al. LogSed: anomaly diagnosis through mining time-weighted control flow graph in logs [C]//2017 IEEE 10th International Conference on Cloud Computing (CLOUD). Honolulu, USA: IEEE, 2017: 447 – 455. DOI:10.1109/CLOUD.2017.64
[7] JIA T, CHEN P F, YANG L, et al. An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services [C]//2017 IEEE International Conference on Web Services (ICWS). Honolulu, USA: IEEE, 2017: 25 – 32. DOI:10.1109/ICWS.2017.12
[8] FU Q, LOU J G, WANG Y, et al. Execution anomaly detection in distributed systems through unstructured log analysis [C]// The 9th IEEE International Conference on Data Mining. Miami Beach, USA: IEEE, 2009: 149 – 158. DOI: 10.1109/ICDM.2009.60
[9] BABENKO A, MARIANI L, PASTORE F. AVA: automated interpretation of dynamically detected anomalies [C]//The 18th International Symposium on Software Testing and Analysis. Chicago, USA: ISSTA, 2009: 237 – 248. DOI: 10.1145/1572272.1572300
[10] YEN T F, OPREA A, ONARLIOGLU K, et al. Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks [C]//The Annual Computer Security Applications Conference. New Orleans, USA: ACM, 2013: 199 – 208. DOI:10.1145/2523649.2523670
[11] ZHAOX, Y.ZHANG, LIOND, et al. lprof: a non-intrusive request flow profiler for distributed systems [C]//Usenix Symposium on Operating System Implementation & Design. Broomfield, USA: OSDI, 2014, 629 – 644
[12] YU X, JOSHI P, XU J W, et al. CloudSeer [J]. ACM SIGPLAN notices, 2016, 51(4): 489 – 502. DOI:10.1145/2954679.2872407
[13] TAK B C, TAO S, YANG L, et al. LOGAN: problem diagnosis in the cloud using log-based reference models [C]//2016 IEEE International Conference on Cloud Engineering (IC2E). Berlin, Germany: IEEE, 2016: 62 – 67. DOI: 10.1109/IC2E.2016.12
[14] AALST WVAN DER, WEIJTERS T, MARUSTER L. Workflow mining: discovering process models from event logs [J]. IEEE transactions on knowledge and data engineering, 2004, 16(9): 1128 – 1142. DOI:10.1109/TKDE.2004.47
[15] LOU J G, FU Q, YANG S Q, et al. Mining program workflow from interleaved traces [C]//Proceedings of the 16th ACM SIGKDD International Conference on Knowledge discovery and data mining. Washington, USA: ACM, 2010: 613 – 622. DOI:10.1145/1835804.1835883
[16] YUAN D, MAI H H, XIONG W W, et al. SherLog [J]. ACM SIGARCH computer architecture news, 2010, 38(1): 143 – 154. DOI:10.1145/1735970.1736038
[17] FU Q, LOU J G, LIN Q W, et al. Contextual analysis of program logs for understanding system behaviors [C]//The 2013 10th Working Conference on Mining Software Repositories (MSR). San Francisco, USA: IEEE, 2013: 397 – 400. DOI:10.1109/MSR.2013.6624054
[18] DU M, LI F F, ZHENG G N, et al. DeepLog: anomaly detection and diagnosis from system logs through deep learning [C]//The 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas, USA: ACM, 2017: 1285 – 1298. DOI:10.1145/3133956.3134015
[19] MENG W B, LIU Y, ZHU Y C, et al. LogAnomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs [C]//The 28th International Joint Conference on Artificial Intelligence. Macao, China: IJCAI, 2019: 4739 – 4745. DOI:10.24963/ijcai.2019/658
[20] LIN Q W, ZHANG H Y, LOU J G, et al. Log clustering based problem identification for online service systems [C]//Proceedings of the 38th International Conference on Software Engineering Companion. Austin, USA: ACM, 2016: 102 – 111. DOI:10.1145/2889160.2889232
[21] SIDDIQUI M A, FERN A, DIETTERICH T G, et al. Feedback-guided anomaly discovery via online optimization [C]//The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. London, United Kingdom: ACM, 2018: 2200 – 2209. DOI:10.1145/3219819.3220083
[22] DAS S, WONG W K, DIETTERICH T , et al. Incorporating expert feedback into active anomaly discovery [C]//16th International Conference on Data Mining (ICDM): IEEE, 2017, 853 – 858
[23] DAS S, WONG W K, FERN A, et al. Incorporating feedback into tree-based anomaly detection [EB/OL]. [2021-01-20]. https://arxiv.org/abs/1708.09441
[24] HE P J, ZHU J M, ZHENG Z B, et al. Drain: an online log parsing approach with fixed depth tree [C]//2017 IEEE International Conference on Web Services (ICWS). Honolulu, USA: IEEE, 2017: 33 – 40. DOI: 10.1109/ICWS.2017.13
[25] GOMEZ RODRIGUEZ M, LESKOVEC J, SCHÖLKOPF B. Structure and dynamics of information pathways in online media [C]//The sixth ACM international conference on Web search and data mining - WSDM'13. Rome, Italy: ACM, 2013: 23 – 32. DOI: 10.1145/2433396.2433402

Biographies

HAN Jing (han.jing28@zte.com.cn) received her master's degree from Nanjing University of Aeronautics and Astronautics, China. She has been with ZTE Corporation since 2000, where she had been engaged in 3G/4G key technologies from 2000 to 2016. She has become a technical director responsible for intelligent operation of cloud platforms and wireless networks since 2016. Her research interests include machine learning, data mining, and signal processing.

JIA Tong is a doctoral researcher at Department of Computer Science and Technology, Peking University, China. His research interests include distributed computing and algorithmic IT operations.

WU Yifan is pursuing his doctorate at School of Software and Microelectronics in Peking University, China. His research mainly focuses on distributed computing and algorithmic IT operations.

HOU Chuanjia is pursuing his master's degree at School of Software and Microelectronics in Peking University, China. His research focuses on algorithmic IT operations.

LI Ying is a researcher at National Engineering Research Center for Software Engineering, Peking University, China. She is also a professor of School of Software and Microelectronics, Peking University. Her research interests include distributed computing and trusted computing.