Behavior Flow Graph Construction from System Logs for Anomaly Analysis

Hang Ling[1], Jing Han[2], Jiayi Pang[1], Jianwei Liu[2], Jia Xu[1], and Zheng Liu[1](✉)

[1] Nanjing University of Posts and Telecommunications, Nanjing, China
{1217043023,b16041505,xujia,zliu}@njupt.edu.cn

[2] ZTE Corporation, Shenzhen, China
{han.jing28,liu.jianweizp}@zte.com.cn

Abstract. Anomaly analysis plays a significant role in building a secure and reliable system. Raw system logs contain important system information, such as execution paths and execution time. People often use system logs for fault diagnosis and root cause localization. However, due to the complexity of raw system logs, these tasks can be arduous and ineffective. To solve this problem, we propose ETGC (Event Topology Graph Construction), a method for mining event topology graph of the normal execution status of systems. ETGC mines the dependency relationship between events and generates the event topology graph based on the maximum spanning tree. We evaluate the proposed method on data sets of real systems to demonstrate the effectiveness of our approach.

Keywords: Event topology graph · System logs · Anomaly detection · Maximum spanning tree

1 Introduction

Systems in business giants such as Google and Amazon generate tens of billions of logs every day. Numerous system logs are of great value in various application fields, and one application domain is to extract valuable knowledge from these system logs for building secure systems [4]. System logs reveal various event characteristics at critical moments, and they contain essential information concerning the operating status of the system, such as the execution traces [9]. Such system logs are the universally available resource among almost all computer systems, which is essential for understanding the overview system status.

Anomaly analysis is indispensable to establish a secure and reliable system. With the rapid iteration of the system version, the system is suffering from cyber-attacks increasingly. However, relying solely on the experience of engineers and domain experts is undoubtedly inefficient and inaccurate. Therefore, log analysis has always been a hot topic in the field of system operation and maintenance, especially in the detection of anomaly events [2,8].

Existing approaches that leverage system logs for anomaly detection can be grouped into two categories: feature-based and workflow-based. Feature-based methods construct feature vectors from system logs and employ Principal Component Analysis (PCA) for anomaly detection [6,11]. Workflow-based approaches build the execution flow graphs based on system logs from normal executions [3,5,12]. Cloudseer [12] models workflow automata by repeating executions of one single task, which analyzes dependencies between events. Tong et al. propose an approach called Logsed [3], which mines the control flow graphs with time weights from operational logs and transaction logs. However, it is a critical task to extract workflow graphs from massive logs.

By carefully considering the characteristics of system logs, we have the following observations: (1) The log format varies with different system platforms, which is usually unstructured. Therefore, it is a challenging task to parse a considerable amount system log accurately and convert them into events. (2) The sequential information of log messages is essential for problem diagnosis. Modern cloud computing platforms execute tasks in parallel and system logs printed by the terminal are often intertwined. Even a single task can perform asynchronous operations which could cause interleaved logs.

The main contributions of this paper are summarized below:

– In this paper, we propose ETGC (Event Topology Graph Construction), an effective approach to mine event topology graph based on the maximum spanning tree from interleaved system logs. Compared with existing anomaly detection approaches, our method can detect more anomalies with high accuracy.
– We evaluate the effectiveness of ETGC mining algorithm on OpenStack data set and BGL data set. The experimental results prove that our method has higher accuracy and interpretability than other algorithms.

The remainder of this paper is organized as follows. In Sect. 2, we explain in detail how to construct event topology graphs, followed anomaly diagnosis in Sect. 3. We conducted extensive experiments and report the results in Sect. 4. Finally, we conclude the paper in Sect. 5.

2 Event Topology Graph Construction for Anomaly Analysis

Our approach consists of an offline phase and an online phase. In the offline phase, we adopt some simple but effective cluster approaches such as BSG [1] to generate log templates of high quality. Then we use these log templates to generate the event topology graphs, which represents the normal execution trace of the system. In the online phase, by comparing the newly arrived execution log sequence and the event topology graphs, we can find their deviation and detect anomaly events.

2.1 Parsing the Logs

Log parsing [1] converts raw and free form system logs into structured log templates with specific formats. A piece of raw system log can be divided into two parts: the constant part and the variable part. The constant part is the constant string from original texts printed by the source codes. The variable part usually carries various status information of the system. Log templates are extracted from raw system logs. For example, if the raw log is "initialize prop version req-f3eaa3dd-321d-44db-b705-937a1c26a01b", then the extracted log template will be "initialize prop version *". Compared with present works mainly based on the templates, we have fully considered the log timestamp information in addition, which significantly improves the accuracy of detecting anomalies.

2.2 Finding Candidate Successor Group

The Computation of the candidate successor group contains two steps: (1) Successor group generation. (2) Noise event filtering. Mining the candidate successor group aims at finding all the possible successor events of reference templates. First, we record all the distinct events in two closest events with the same reference template and then add them to the candidate successor group of the reference template. The noise event filtering aims to keep meaningful but rare events. We use an adaptive correlation probability to filter out noise while retaining normal rare events. Let us denote the probability of occurrence of predecessor template A as P_A, the probability of occurrence of successor template B as P_B, and the occurrence times of template B in the successor group of A as $N_{(A|B)}$. We compute the correlation possibility between event A and event B in the successor group.

$$SUP_{(A|B)} = \frac{N_{(A|B)}}{min(P_A, P_B)} * sigmoid(min(P_A, P_B)). \tag{1}$$

The threshold for SUP should be small enough to filter noise event and retain rare and meaningful events. After we set a filtering threshold, if the correlation probability is larger than the threshold, then event B is added to the final successor group of A.

2.3 Mining Dependent Event Pair

The existing method usually uses a time window mechanism to retain the subsequent events of the reference templates in a statistical manner. The items in the same sliding window are considered to be subsequent events of the same reference template. However, it is difficult to estimate the length of the time window accurately.

Li and Ma [10] proposed to use several statistics for detecting dependent event pairs and filtering candidate event pairs. The sequence of points of event type A is denoted as $P_A = < a_1, a_2, ..., a_m >$, and a_i is the specific timestamp of one log entry of event type A. Assume that the time range of the point sequence P_A is $[0, T]$, given a point z, the minimum positive distance between the z and sequence P_A is defined as

$$d(z, P_A) = min||x - z||, x \in P_A, x \geq z. \tag{2}$$

The unconditional distribution of waiting time of event B is

$$F_B(r) = P(d(z, P_B)) \leq r, \tag{3}$$

where r is a real number. The conditional distribution of waiting time of event B with respect to event A is

$$F_{B|A}(r) = P(d(z, P_B)) \leq r, z \in P_A, \tag{4}$$

where r is a real number, z is a point of P_A in the point sequence, and $F_{B|A}$ describes the conditional probability in the case of event A at time z. Then we have the following definition [10].

Definition 1. *[10] Given two corresponding sequence of points for event types A and B, if $F_B(r)$ and $F_{B|A}$ are significantly different, then statistically, event B is considered to be dependent on event A. Specifically, the dependency test between events A and B can be compared by $F_{B|A}$ and $F_B(r)$. Assuming that A and B are independent of each other, according to the central limit theorem*

$$Z = \frac{M_B - M_{B|A}}{\sqrt{\frac{var(F_B(r))}{m}}} \sim N(0, 1), \tag{5}$$

where $var(F_B(r))$ represents the variance of $F_B(r)$, M_B and $M_{B|A}$ represent the first moment of $F_B(r)$ and $F_{B|A}(r)$ respectively.

2.4 Discovering Transition Time Lag

Existing methods tend to use a fixed time weight for edges in the event topology graph, which indicates the transition time period between adjacent events. However, in real world systems, fixed time lags are not practical due to noise interference, unsynchronized clocks, and so forth. Time lags usually fluctuate within a range.

For each dependent event pair $< T_i, T_j >$, we look into two adjacent template T_i and T_j in the log stream and record all the time period as $< t_1, t_2, ..., t_m >$. We use a time distribution $f(t)$ to describe the transition time of event pair $< T_i, T_j >$. Since the time distribution represents the time lag sequence of event pair, we propose a cluster-based method to get rid of redundant event pairs. In this method, we divide the time-delay sequence into multiple time lag clusters. Then the maximum and minimum values of the clusters are considered as the boundary values of the transition time interval. Intuitively, we perform the chi-square test on these time lag clusters. If they pass the chi-square test, then the event pairs are considered to have the dependency relationship, and the time delay interval is used as the time lag interval of the event pair.

2.5 Generating Maximum Spanning Tree

In the graph theory, a spanning tree of a graph is a subgraph that contains all the vertices and the maximum spanning tree [7] is the minimum connected graph with maximum weight. In a maximum spanning tree, each node represents a single distinct event, and weights of connected edges represent the transition probability between predecessor and successor events. Successor events are not always immediately follow reference events, so that some structures like loop structures and detour structures may be missing from the event topology. However, the spanning tree represents the backbone of the entire workflow. Even if some meaningful structures are missing, we can still retrieve them through the original dependency relationship between events. We employ the attenuation factor to control the possibility of the existence of the detour structure. We first define the step size as the distance from the starting node to the terminal node in the maximum spanning tree structure. Next, we define the probability of the existence of the detour structure between the starting node and the terminal node as:

$$d(E_1, E_2) = log(1 + path(E_1, E_2)), \tag{6}$$

where the $path(E_1, E_2)$ refers to the step size between E_1 and E_2 in maximum spanning tree. Then we could set a suitable threshold to preserve the edge between $E1$ and $E2$.

3 Anomaly Diagnosis

There are two kinds of anomalies: event anomalies and time anomalies. An event anomaly is raised when an unexpected log entry occurs, which cannot match to any node in the event topology graph. Unexpected log entry indicates an abnormal event that cannot be matched with any log template or a redundant occurrence of a log template.

A time anomaly is raised when a child node of a parent node occurs, but the interval time is not within the time lag interval. The time lag interval records the maximum transition time and minimum transition time of the event. Any transition time that occurs within this time lag interval is considered as a normal event. Hence, this type of anomaly is more instantaneous and could be captured easily.

4 Experimental Evaluation

4.1 The Datasets

We evaluate our approach ETGC through two real log datasets. Detailed information about the two datasets is as followed:

1. OpenStack cloud platform log dataset: OpenStack, a cloud computing platform based on PaaS, provides cloud service for millions of people all over the world and its logs are accessible to users. This data includes 30 normal deployments and 3 abnormal deployments, each of which is related to the deployment of the cluster. We collected the data from the two components: cf-pdman and pdm-cli.

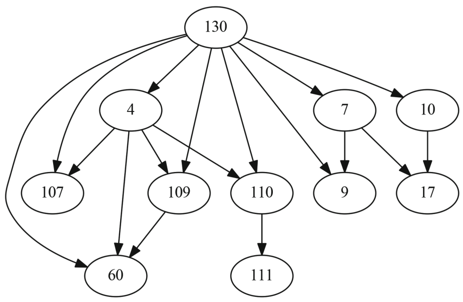

Fig. 1. An event topology graph from the OpenStack dataset

2. BGL log dataset: Blue Gene/L supercomputer system log dataset contains 4,747,963 logs, where 348,460 entries are labeled as anomalies. We choose this dataset because it contains many log templates which only appear during a certain time period.

4.2 A Case Study

To illustrate our method of anomaly diagnosis, we take an event topology graph for example. Figure 1 is a part of the event topology graph generated from OpenStack dataset. In this figure, each node represents a distinct event, and the weight of each edge represents the transition probability of an event pair. Moreover, Table 1 provides further information about these log templates. For example, path (130, 7, 17) in the transaction flow diagram in Fig. 1 represents the configuration of attributes on the server node.

4.3 Anomaly Diagnosis Evaluation

Figures 2 and 3 demonstrate the Error Event Pair Percent and the Average Error Event Percent in normal deployments and abnormal deployments results, respectively. Whether in abnormal deployments such as 514, 17, 60 or normal deployments, ETGC can detect more Error Event Pair than Logsed. The left side and right side of Fig. 4 show the proportion of event pairs on BGL data set detected by ETGC and Logsed, respectively. The result demonstrates that ETGC can detect more normal and abnormal event pairs than Logsed, which proves that our approach is feasible.

Table 1. Event IDs and the corresponding events

Event IDs	Events
130	sqlalchemy.orm.relationships.RelationshipProperty Node.servers local/remote pairs [(nodes.server_id/servers.id)] *
4	sqlalchemy.orm.mapper.Mapper (Server—servers)_configure_property(node_list, RelationshipProperty) *
107	sqlalchemy.orm.relationships.RelationshipProperty Node.servers secondary synchronize pairs *
109	sqlalchemy.orm.relationships.RelationshipProperty Server.Node secondary synchronize pairs *
60	sqlalchemy.orm.mapper.Mapper (Node—nodes) initialize prop created_at *
110	sqlalchemy.orm.mapper.Mapper (Node—nodes) initialize prop timestamp *
111	sqlalchemy.orm.relationships.RelationshipProperty Server.node_list secondary synchronize pairs *
7	sqlalchemy.orm.mapper.Mapper (Node—nodes) _configure_property(servers, RelationshipProperty) *
9	sqlalchemy.orm.mapper.Mapper (CtrlSwitch—ctrl_switch)_configure_property(created_at, Column) *
10	sqlalchemy.orm.mapper.Mapper (Lock—locks) _configure_property(state, Column) *
17	sqlalchemy.orm.mapper.Mapper (PsmRole—psmroles) _configure_property(id, Column) *

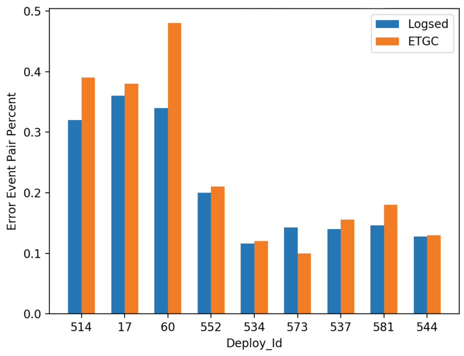

Fig. 2. The OpenStack deployments error event pair percent

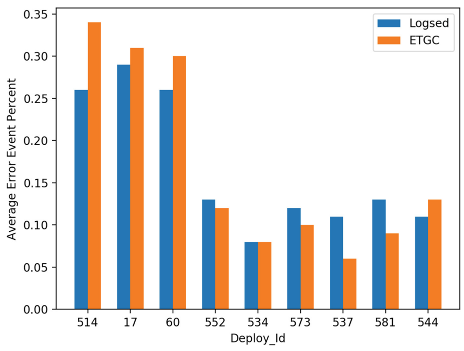

Fig. 3. OpenStack deployments average error event percent

4.4 The Execution Time

At the first stage, we study the time it takes to generate candidate event pairs. At the second stage, we pay attention to the time it costs for our algorithm to filter these candidate event pairs and generate the event topology graph. In Fig. 5, the solid line and dotted line refer to the time spent at the first and second stage. It shows that the time taken to generate candidate event pairs

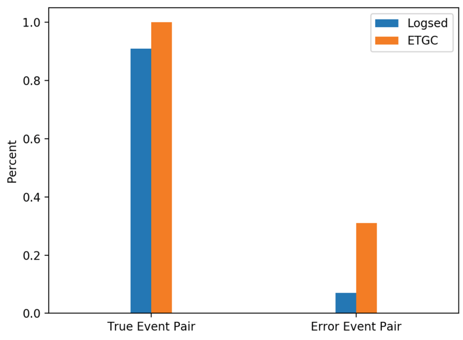

Fig. 4. The BGL validation result

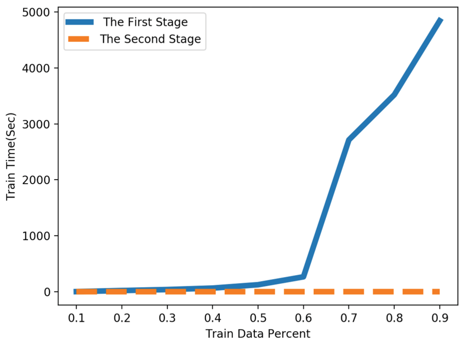

Fig. 5. Execution time on BGL dataset

grows exponentially as the number of data increases. To solve this problem, we can use the multi-thread program to reduce the time spent at the first stage of our approach.

5 Conclusion

In this paper, we proposed ETGC (Event Topology Graph Construction), an effective approach to diagnose the abnormal events based on system logs. By using the maximum spanning tree generation, ETGC constructs the meaningful event topology graphs based on dependent event pairs. Evaluation results show that our approach can achieve superior performances in anomaly event detection.

Acknowledgments. This work is supported in part by Nanjing University of Posts and Telecommunications under Grant No. NY215045 and NY219084, and Shanghai Sailing Program under Grant No. 18YF1423300.

References

1. Guo, S., Liu, Z., Chen, W., Li, T.: Event extraction from streaming system logs. Inf. Sci. Appl. **2018**, 465–474 (2019)
2. He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: system log analysis for anomaly detection. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 207–218, October 2016
3. Jia, T., Yang, L., Chen, P., Li, Y., Meng, F., Xu, J.: LogSed: anomaly diagnosis through mining time-weighted control flow graph in logs. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), pp. 447–455 (2017)
4. Li, T., Liu, Z., Zhou, Q.: Application-driven big data mining. ZTE Technol. J. **22**(2), 49–52 (2016)
5. Lin, Q., Zhang, H., Lou, J., Zhang, Y., Chen, X.: Log clustering based problem identification for online service systems. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 102–111 (2016)

6. Lou, J.G., Fu, Q., Yang, S., Xu, Y., Li, J.: Mining invariants from console logs for system problem detection. In: USENIX Annual Technical Conference, pp. 1–14 (2010)
7. McDonald, R., Pereira, F., Ribarov, K., Hajič, J.: Non-projective dependency parsing using spanning tree algorithms. In: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing, pp. 523–530 (2005)
8. Nagaraj, K., Killian, C., Neville, J.: Structured comparative analysis of systems logs to diagnose performance problems. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, p. 26 (2012)
9. Tak, B.C., Tao, S., Yang, L., Zhu, C., Ruan, Y.: Logan: problem diagnosis in the cloud using log-based reference models. In: 2016 IEEE International Conference on Cloud Engineering (IC2E), pp. 62–67, April 2016
10. Li, T., Ma, S.: Mining temporal patterns without predefined time windows. In: Fourth IEEE International Conference on Data Mining (ICDM 2004), pp. 451–454, November 2004
11. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.I.: Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, pp. 117–132 (2009)
12. Yu, X., Joshi, P., Xu, J., Jin, G., Zhang, H., Jiang, G.: Cloudseer: workflow monitoring of cloud infrastructures via interleaved logs. ACM SIGPLAN Not. **51**(4), 489–502 (2016)