System Problem Detection by Mining Process Model from Console Logs

Jian Li, Jian Cao

System Problem Detection by Mining Process Model from Console Logs

Jian Li and Jian Cao*

Department of Computer Science and Engineering,
Shanghai Jiao Tong University,
Shanghai 200240, China
jian-li@sjtu.edu.cn,cao-jian@sjtu.edu.cn

Abstract. Given the explosive growth of large-scale services, manually detecting problems from console logs is infeasible. In the current study, we propose a novel process mining algorithm to discover process model from console logs, and further use the obtained process model to detect anomalies. In brief, the console logs are first parsed into events, and the events from one single session are further grouped to event sequences. Then, a process model is mined from the event sequences to describe the main system behaviors. At last, we use the process model to detect anomalous log information. Experiments on Hadoop File System log dataset show that this approach can detect anomalies from log messages with high accuracy and few false positives. Compared with previously proposed automatic anomaly detection methods, our approach can provide intuitive and meaningful explanations to human operators as well as identify real problems accurately. Furthermore, the process model is easy to understand.

Keywords: anomaly detection, process mining, logs, system monitoring

1 Introduction

Traditionally, operators inspect the console logs manually by searching for keywords such as "error" or "exception". But it has been shown to be infeasible due to couple of reasons. First, modern systems are large scale, and are generating huge logs everyday. Thus, its too difficult to manually identify the real problems from tons of data. Second, the large-scale modern systems are too complex for one single developer to understand, and it makes it a great challenge for anomaly detection from huge console logs. Third, fault tolerant mechanisms are usually employed in large-scale systems. Hence, keywords like "error" or "exception" don't necessarily indicate real problems.

Recently, several automatic anomaly detection methods based on log analysis have been proposed. For example, Lin et al. [5] proposed a clustering based method to detect the abnormal log messages. Xu et al. [8] detect anomalies

* Corresponding author

using Principal Component Analysis(PCA). Process mining [7] is a technique to distill a structured process description from a set of real executions. In this work, we proposed a bottom-up process mining method to discover the process model of the main system behaviors based on console log information. If a new log breaks certain process model, we say it is anomalous.

As the process model is intuitive with meaningful information, our approach can not only automatically detect system anomalies but also provide meaningful interpretation for problem diagnosis.

2 Process Modeling Notation

There are various process modeling notations, such as Petri Nets, Workflow Nets, BPMN and YAWL. Although they are quite different in notations, it is relatively easy to translate the process model from one notation to another. In the present work, a variant of process tree is defined to describe the models mined from log information using our algorithm.

2.1 Process Trees Variant

Definition 1 (Process Tree).
Let A be a finite set of activities. Symbol $\tau \notin A$ denotes the silent activities. $\bigoplus = \{\rightarrow, \times, \wedge_{(m,n)}, \circlearrowleft_{(m,n)}\}$ *is the set of process tree operators.*

- *If $a \in A \cup \{\tau\}$, then $Q = a$ is a process tree,*
- *If $Q_1, Q_2, \ldots, Q_n$ with $n > 0$ are process trees and $\oplus$ is a process tree operator, then $\oplus(Q_1, Q_2, \ldots, Q_n)$ is a process tree.*

Definition 2 (Process Tree Operators).
Let Q be a process tree over A. $L(Q)$ is the set of traces that can be generated by Q. $\diamondsuit(L(Q_1), L(Q_2), \ldots, L(Q_n))$ generates the set of all interleaved sequences. $L(Q)$ is defined recursively:

- $L(Q) = \{[a]\}$ *if* $Q = a \in A$,
- $L(Q) = \{[]\}$ *if* $Q = \tau$,
- $L(Q) = \{[a_1, a_2, \ldots, a_n] | a_i \in L(Q_i), \forall i \in 1 \ldots n\}$ *if* $Q = \rightarrow (Q_1, Q_2, \ldots, Q_n)$,
- $L(Q) = \{[a_i] | a_i \in L(Q_i), \forall i \in 1 \ldots n\}$ *if* $Q = \times(Q_1, Q_2, \ldots, Q_n)$,
- $L(Q) = \diamondsuit(\diamondsuit_1(L(Q_1), L(Q_2), \ldots, L(Q_s)),$
  $\diamondsuit_2(L(Q_1), L(Q_2), \ldots, L(Q_s)), \ldots,$
  $\diamondsuit_u(L(Q_1), L(Q_2), \ldots, L(Q_s)))$ *with* $u \in m, \ldots, n$ *if* $Q = \wedge_{(m,n)}(Q_1, Q_2, \ldots, Q_s)$,
- $L(Q) = \{[a_{11}, a_{12}, \ldots, a_{1s}, \ldots, a_{t1}, a_{t2}, \ldots, a_{ts}] | a_{ij} \in L(\rightarrow (Q_1, Q_2, \ldots, Q_s)), \forall j \in 1 \ldots s, \forall i \in 1 \ldots n, t \in m \ldots n\}$ *if* $Q = \circlearrowleft_{(m,n)} (Q_1, Q_2, \ldots, Q_s)$
- $\wedge$ *is the short form of* $\wedge_{(1,1)}$, $\circlearrowleft$ *is the short form of* $\circlearrowleft_{(1,1)}$

3 Proposed Approach

Our approach consists of three main steps: log parsing, process mining, and anomaly detection.

3.1 Log parsing

Usually raw log messages are difficult to be directly processed by computers as they are unstructured. In this work, log templates are first extracted from unstructured log messages. Log messages with the same log template are grouped to the same type of event. Then the events within the same session are converted to a single sequence according to the recorded time. A sequence of events in time order is called an event trace.

3.2 Process mining

Next, a three-phase process mining algorithm is utilized to uncover the process model which can represent main system behaviors. This approach can discover four kinds of basic control flow structures, which are sequence, choice, loop, and concurrency.

1. Discover subroutines
   A subroutine is basically a unit that contains a sequence of program instructions to perform a specific task in computer programming. Subroutines usually lead to groups of events with certain patterns in event traces. We use a statistical based method to identify the set of events that correspond to the subroutine, the structure of the events within the set, and how subroutines are called (in a roll or parallelly).
2. Discover the Main Control Flow
   In the previous step, the original events that correspond to subroutines are replaced by new combined events that represents subroutines. By taking each event in a trace as a node, and two adjoining events as two nodes connected by an edge, then each event trace is a directed acyclic graph. The directed graph was used to discover the main control flows of our target system.
3. Adjust the Model
   The process tree model mined as described above is constructed by two nodes each time iterating step by step. To make the model more concise, we do some adjust on the process tree representation of model without changing the semantic meanings in this step.

3.3 Anomaly detection

At last, the discovered process model is applied to detect system anomalies. If an observed event sequence conforms to the process model, it will be labeled as normal. Otherwise, the ones which violate the process model are labeled as anomalies.

Our algorithm which checks whether a event trace conforms to the process tree model runs in a recursive manner. If the tree has only one node, then the conformance can be checked easily. Otherwise, we check the conformance of each subtree first, and then check whether the event trace conforms to the root node's rule. Every subtree contains a set of events. For each subtree, a sub-trace that

only contains the corresponding events was extracted from the original event trace to check the conformance.

The process model mined by our approach keeps the patterns of event traces which are generated by the main system behaviors. The model depicts system execution paths in a tree structure and is easy to understand.

4 Experiments

We use HDFS log dataset [8] to evaluate the performance of our approach with the permission of the authors. HDFS dataset contains 11,175,629 log messages in total. All these log messages belong to 575,061 sessions. Among them, 16,838 sessions are manually labeled as anomalies by experts. Fig. 1 shows the process model mined from HDFS logs. To evaluate the accuracy of our approach, we use three commonly used metrics: precision, recall, and f-measure.

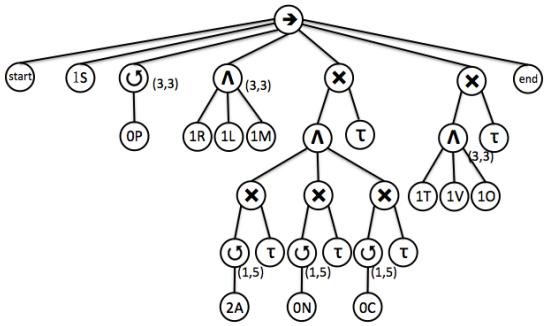

Fig. 1. Process Model of HDFS logs

He et al. [4] evaluated six state-of-the-art log-based anomaly detection methods. Among these methods, Log Clustering [5], PCA [8] and Invariant Mining [6] are unsupervised methods. We repeated their experiments and got similar results. Fig. 2(a) shows the results of our approach and other three unsupervised methods on HDFS data. Our approach achieved the recall of 100% while obtain high detection precision of 89%. To evaluate the stability of our method, the dataset are first split into ten subsets and we perform our method on each of them. The results are shown in Fig. 2(b). Our approach detects anomalies by constructing a model that depicts the main system behaviors. Therefore it is not sensitive to the noises in the data.

5 Related Work

Considerable research efforts have been conducted on anomaly detection. Chandola et al. [2, 3] classified anomalies into three categories (point anomalies,

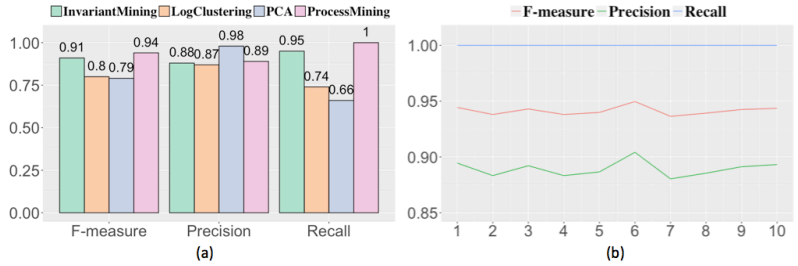

Fig. 2. Results

contextual anomalies and collective anomalies) and compared various kinds of anomaly detection techniques. Analyzing console logs for system problem detection has been an active research area. Xu et al. [8] first extract message count vectors from logs, and then detect anomalies using Principal Component Analysis(PCA). Lou et al. [6] detect anomalies using invariants mined from console logs. Clustering technique [5] and other machine learning techniques [1] have been applied to detect anomalies.

Acknowledgments. This work is supported by China National Science Foundation (Granted Number 61472253).

References

1. Alonso, J., Belanche, L., Avresky, D.R.: Predicting software anomalies using machine learning techniques. In: Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on. pp. 163–170. IEEE (2011)
2. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR) 41(3), 15 (2009)
3. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection for discrete sequences: A survey. IEEE Transactions on Knowledge and Data Engineering 24(5), 823–839 (2012)
4. He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: System log analysis for anomaly detection. In: Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on. pp. 207–218. IEEE (2016)
5. Lin, Q., Zhang, H., Lou, J.G., Zhang, Y., Chen, X.: Log clustering based problem identification for online service systems. In: Proceedings of the 38th International Conference on Software Engineering Companion. pp. 102–111. ACM (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 (2010)
7. Van Der Aalst, W., Adriansyah, A., De Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, J.C., van den Brand, P., Brandtjen, R., Buijs, J., et al.: Process mining manifesto. In: International Conference on Business Process Management. pp. 169–194. Springer (2011)
8. 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. ACM (2009)