Lopper: An Efficient Method for Online Log Pattern Mining Based on Hybrid Clustering Tree

Jiawei Liu[1], Zhirong Hou[1], and Ying Li[2(✉)]

[1] School of Software and Microelectronics, Peking University, Beijing, China
{colordown,hou.zhirong}@pku.edu.cn
[2] National Engineering Center of Software Engineering, Peking University, Beijing, China
li.ying@pku.edu.cn

Abstract. Large-scale distributed system suffers from the problem that system manager can't discover, locate and fix system anomaly in time when system malfunctions. People often use system logs for anomaly detection. However, manually inspecting system logs to detect anomaly is unfeasible due to the increasing scale and complexity of distributed systems. As a result, various methods of automatically mining log patterns for anomaly detection have been developed. Existing methods for log pattern mining have drawbacks of either time-consuming or low-accuracy. In order to address these problems, we propose Lopper, a hybrid clustering tree for online log pattern mining. Our method accelerates the mining process by clustering raw log data in one-pass manner and ensures the accuracy by merging and combing similar patterns with different kernel functions in each step. We evaluate our method on massive sets of log data generated in different industrial applications. The experimental results show that Lopper achieves the accuracy with 92.26% on average which is much better than comparative methods and remains high efficiency at the same time. We also conduct experiments on system anomaly detection task using the log patterns generated by Lopper, the results show an average F-Measure performance of 91.97%, which further proves the effectiveness of Lopper.

Keywords: Log analysis · Log pattern mining · System anomaly detection · Data structure

1 Introduction

Due to the complex composition and operation logic of large-scale distributed system, it is often difficult for system managers to discover, locate, and diagnose system anomaly in the event of a system failure. How to quickly and accurately identify, detect and even predict the failure of large-scale distributed system has become an important research issue.

At present, the mainstream technology of anomaly diagnosis for large-scale distributed systems is based on system logs [1, 2, 4, 6, 15]. System logs based anomaly detection has following advantages: 1. System logs can track program execution logic

S. Hartmann et al. (Eds.): DEXA 2019, LNCS 11706, pp. 63–78, 2019.
https://doi.org/10.1007/978-3-030-27615-7_5

and capture exceptions across components and services. 2. System logs describe the system state more fine-grained which can locate specific error log, event information and even program code.

System logs are used to record the system runtime information. In general, system logs are consisted of constants and variables. The constant is used to describe the role of current code or the reason for triggering log print statement and the variable reflects the dynamic system runtime data, such as IP address and parameter information [4, 6].

Researchers often focus on the constant part when using system logs for anomaly detection [3, 4, 6]. To better describe the work, we come up with the concept of log pattern: **A log pattern** [3, 13] is composed of the constant part of a log record and wildcards which are used for replacing variables. Moreover, a log pattern is used to identify a specific type of system log and essentially corresponds to a log print statement in program code. **Log pattern mining** [8, 9, 13] refers to the process of mining log patterns from massive raw log data. The following example shows a set of original system log records and their corresponding log patterns (see Fig. 1).

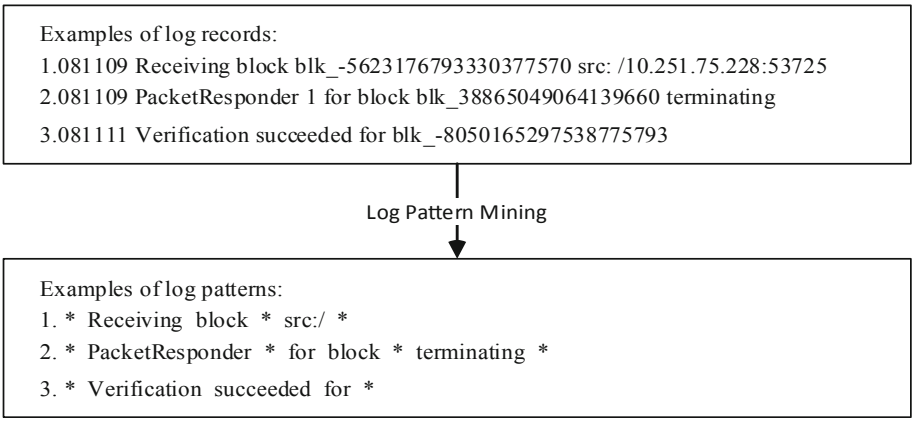

Fig. 1. Examples of log records and log patterns

The research of log pattern mining has been a hotspot for many years, and there have been many research results (related works will be further discussed in detail in Sect. 2). Vaarandi [12] design a data clustering algorithm which learns the idea from the classic Apriori algorithm for mining frequent item sets. However, this method couldn't perform well on large data set due to its high time complexity. Xu et al. [6] propose a method using abstract syntax tree for mining log pattern based on source code, which could be useful but only apply to the condition where source code is accessible. Tang et al. [10] come up with a model named LogSig for clustering log data according to their longest common sequence (LCS), which is not efficient because the process of calculating the LCS of two log records is time-consuming. Makanju et al. [8] design IPLoM which clusters system logs using iterative partitioning, but their method has high space complexity. All the early approaches for log pattern mining act in offline mode and we observe that they are not accurate enough, with the increasing demand for

real-time analysis of online log data, online log pattern mining with high accuracy [3, 7, 9, 14] has become a trend and necessity.

Existing methods have drawbacks of either time-consuming or low-accuracy [2, 3]. In this paper, we propose an online log pattern mining method named Lopper to improve the performance of log pattern mining task. Lopper is a hybrid clustering tree with three layers, the first of which is used for clustering raw log data, the second is for merging similar clusters and the last is for combining candidate patterns to get final patterns. Lopper accelerates the mining process by clustering raw log data in one-pass manner and ensures the accuracy by merging and combing similar patterns with different kernel functions in each step. We evaluate our method on massive sets of log data generated in different industrial applications, the experimental results show that Lopper achieves the accuracy with 92.26% on average which is much better than comparative methods and remains high efficiency at the same time. We also conduct experiments on system anomaly detection task using the log patterns generated by Lopper, the results show an average F-Measure performance of 91.97%, which further proves the effectiveness of Lopper.

The rest of the paper is organized as follows: Sect. 2 introduces related work on log pattern mining, Sect. 3 presents our online log pattern mining method, Lopper. In Sect. 4, we evaluate the performance of Lopper on massive log data sets and Sect. 5 concludes the paper.

2 Related Work

In this section, we will give a detailed survey on the research of log pattern mining.

2.1 Offline Log Pattern Mining

As we discussed before, Log pattern mining technology refers to the process of mining log patterns from massive raw log data. According to specific technical approach, log pattern mining technology can be classified as following: clustering based methods, frequent item sets mining based methods, static code analysis based methods and other methods.

Clustering based log pattern mining technology [4, 7, 10, 13] introduces clustering algorithm into log pattern mining. The core idea is using the similarity between log records to calculate the distance for clustering. The clustering results are extracted to form the final log patterns. Clustering based method is the mainstream method in log pattern mining technology. This method usually goes as following: 1. Data preprocessing: Using regular expressions based on domain knowledge to replace commonly-used variables in log records with wildcards, such as IP address and timestamp. Data preprocessing can reduce the complexity of following work effectively. 2. Clustering log records: Clustering log data using clustering algorithms based on the distance between log records, such as edit distance [16]. 3. Log pattern extraction: Clusters are formed after clustering and different clusters correspond to different log patterns. Pattern extraction is the process of extracting one certain log pattern from a cluster.

Frequent item sets mining based method [12] is first used in the field of log pattern mining, this method can discover association rules between data and give frequent item sets. Log data has the feature that constants come up more frequent than variables [2, 3], hence by mining the frequent item sets in log records we can get the log patterns. Taking the classic Apriori algorithm as an example, each log record is treated as a transaction and each token (constant or variable) in the log record is treated as a commodity. The frequent item sets mined are often the combination of log constants, which can be reorganized to form log patterns.

Log pattern mining based on static code analysis [6] is aimed at the system whose source code is available. This approach differs significantly from the previous methods because it analyzes the source code instead of log data. The core idea of this method is directly mining log print statement from the source code and generating log patterns.

Other research includes that Zhu et al. [5] treat logs as natural language from the perspective of Natural Language Processing (NLP) and propose an incremental learning model to study the schemes of log data. Cheng et al. [11] design a deep convolutional neural network for automated classification of anomalous events detected from the distributed system logs. Lu et al. [1] come up with a CNN-based model which can automatically learn event relationships in system logs and detect system anomaly.

2.2 Online Log Pattern Mining

All the early approaches for log pattern mining act in offline mode, but with the increasing demand for real-time analysis of online log data, online log pattern mining has become a trend and necessity. Mizutani [9] design the first online log pattern mining model named SHISO, which is a tree with predefined rule for clustering log data in online way. SHISO has to update each time when a new log record comes in, so the model is way too cumbersome. Du et al. [14] propose an online streaming method Spell, which is based on longest common subsequence to parse system logs. This approach is not efficient because the time complexity of this algorithm is O (n^2). He et al. [3] design Drain, an online log parsing approach with fixed depth tree. Drain is more efficient compared to SHISO [9], but Drain firmly relies on the preprocessing of log data and the rules for constructing Drain are arbitrary so that Drain is not widely applicable. Hamooni et al. [7] propose LogMine, which adopts a one-pass algorithm for clustering with high efficiency. But the merging step in LogMine is based on Smith–Waterman Algorithm [17] which is time-consuming.

3 Methodology

In this section, we will introduce our online log pattern mining method, Lopper. Lopper is named on the basis of **lo**g **p**attern min**er**, and the word "lopper" itself has the meaning of a worker who prunes redundant trees, so we expect Lopper to be a superior "lopper" who can finish the job of pruning log data with high accuracy and efficiency. We will give a brief overview of our method first and then explain in detail each step of the mining process.

3.1 Definition

We will first define some terms for the purpose of clearer presentation. System logs are used to record the system runtime information. In general, system logs are consisted of constant and variable. The **Constant** is used to describe the role of the current code or the reason for triggering log print statement and the **Variable** reflects the dynamic system runtime data, such as IP address and parameter information. Either constant or variable is described as a **Token**, so one log record is actually a series of tokens. **A log pattern** is composed of the constant part of a log record and wildcards which are used for replacing variables [2–4, 6, 13].

To be clear, we define $\boldsymbol{T_n^C}$ which means the n-th token in a log record (or log pattern) is a constant and $\boldsymbol{T_n^V}$ which means the n-th token in a log record (or log pattern) is a variable. The **Length** of a log record (or log pattern) is the number of tokens that form the log record (or log pattern).

For example, the log record {Receive 120 KB from 10.0.2.17} has length of 5 and can be formally expressed as $\left\{T_1^{C1}\, T_2^{V1}\, T_3^{C2}\, T_4^{C3}\, T_5^{V2}\right\}$, and the corresponding log pattern {Receive * KB from *} also has length of 5 and can be formally expressed as $\left\{T_1^{C1}\, T_2^{V}\, T_3^{C2}\, T_4^{C3}\, T_5^{V}\right\}$. As we can see from the difference between two expressions, all variables in log pattern are replaced by wildcards because we give equal treatment to variables in log pattern.

3.2 Model Overview

The structure of Lopper is illustrated in Fig. 2. Log records will be first preprocessed using regular expressions based on domain knowledge and commonly-used variables in log records such as IP address and timestamp should be replaced by wildcards in this step. All preprocessed log data will be then sorted into different groups according to their length, and each separate group is an aggregation of log records with identical length.

One-pass clustering will take place in each individual group and generate clusters with different ID. In order to further accelerate the speed of clustering process, we adopt a relatively simple similarity measure function for one-pass clustering, so the preliminary clustering results are not accurate enough. To solve this problem, we will merge similar clusters employing more precise similarity measure function right after the one-pass clustering, hence similar clusters in the same group are merged to form the candidate log patterns. It's possible that candidate log patterns from different groups actually belong to same log pattern, therefore we need to combine similar candidate log patterns across groups to generate the final log patterns. To better demonstrate the log pattern mining process intuitively, we give a running example in Fig. 3.

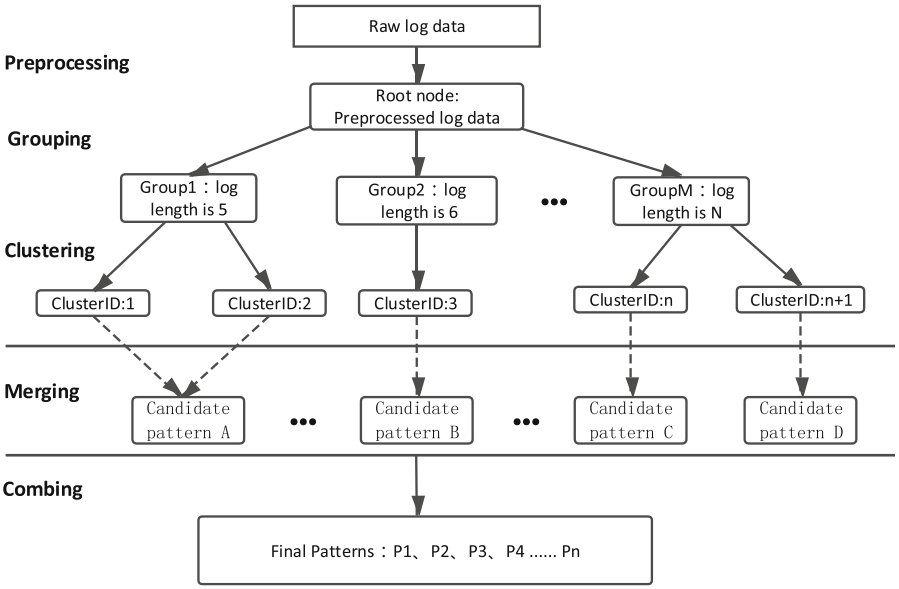

Fig. 2. Structure of Lopper

3.3 Preprocessing and Grouping Log Data by Length

Log pattern mining refers to the process of mining log patterns from massive raw log data, in which variables are distinguished and replaced while constants are extracted to form the final patterns. In fact, preprocessing plays the same role. In this step we use simple regular expressions to parse some commonly-used variables (e.g., IP address, timestamp and number). As shown in Fig. 3, numbers and IP addresses in log records are replaced by wildcards. Preprocessing reduces the complexity of following work to a certain extent. The more pre-defined rules are set up in this step, the more convenient later work will be. But there is an obvious problem that we can't always know the exact content and structure of the log data we are going to parse, hence just relying on preprocessing is unrealistic. In order to make our model widely adaptable, we only adopt limited pre-defined rules in Lopper.

After preprocessing, we then sort all log records into different groups according to their length. We do this for the following reasons: 1. Based on previous research work [2, 15] and our statistical analysis of log data, we have the assumption that log records belong to the same patterns always share identical log length; 2. It will be more convenient and more precise to design similarity measure function if log records in one group are of identical log length; 3. Previous work [7, 8, 13] processes all log data with unequal length at one time and then tries to align log records in order to extract log patterns. By classifying log records at the very start, there is no need for alignment operation which will greatly raise the mining speed.

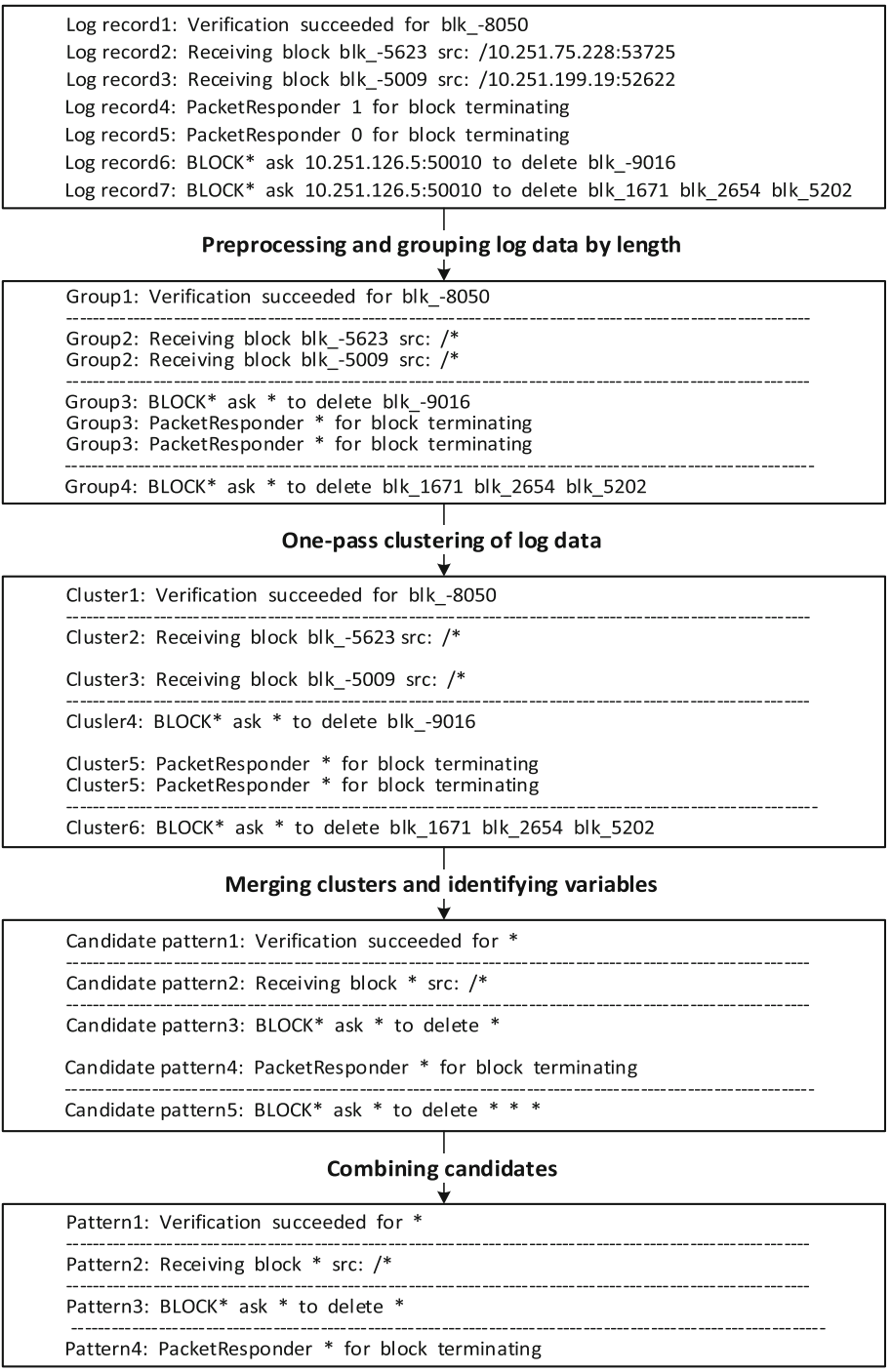

Fig. 3. Running Example of log pattern mining by Lopper

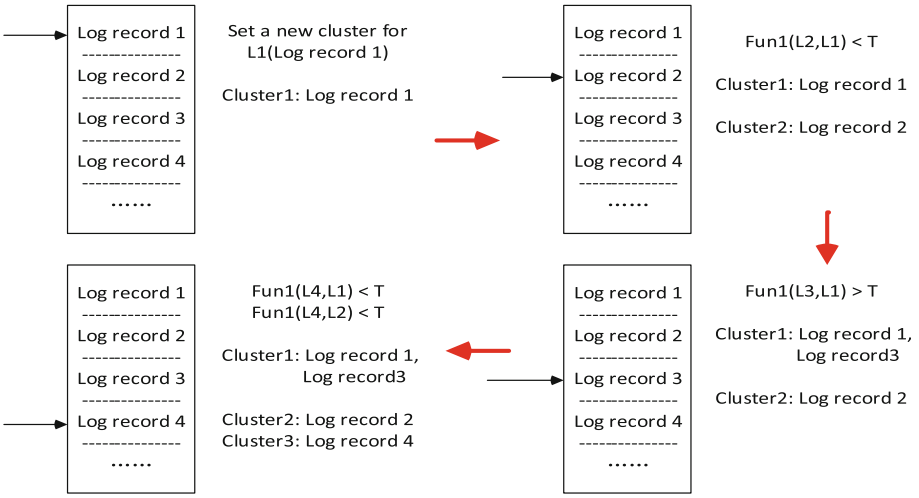

Fig. 4. Running example of one-pass clustering algorithm

3.4 One-Pass Clustering of Log Data

In order to cluster log data with high efficiency, we adopt one-pass clustering algorithm which cluster log data in streaming way. Before describing the algorithm in detail, we first give our similarity measure function. The similarity measure function ***Fun1*** between two log records is defined as following:

$$Fun1(log1, log2) = \frac{\sum_{i=1}^{n} S1(log1(i), log2(i))}{\mathrm{n}} \tag{1}$$

Where $log1$ and $log2$ stand for two separate log records; $log1(i)$ or $log2(i)$ represents the i-th token in this log record; n is the log length of $log1$ (the length of $log1$ and $log2$ is the same); Function $\boldsymbol{S1}(\boldsymbol{x},\boldsymbol{y})$ is used to determine whether two tokens are the same which is defined as following:

$$S1(x, y) = \begin{cases} 1, \ if \ x = y \\ 0, \ otherwise \end{cases} \tag{2}$$

Where x and y are two tokens in log1 and log2 respectively. $\boldsymbol{Fun}1$ reflects the ratio of identical tokens of two log records. We will set a similarity threshold T, if $\boldsymbol{Fun}1(log1, log2) > \mathrm{T}$, we consider the two log records as similar records and put them into the same cluster otherwise they will be sorted into different clusters.

One-pass clustering algorithm starts from the first log record and goes through all log records in streaming way. When the very first log record L1 arrives, we create a new cluster for L1 and L1 is the only log record in this cluster. And then comes the second log record L2, we will calculate the similarity of L1 and L2 using ***Fun1***, if $\boldsymbol{Fun}1(L1, L2) > \mathrm{T}$, we send L2 into the same cluster with L1, otherwise we will create

a new cluster for L2 and L2 will be the only log record in the new cluster. When a new log record Ln arrives, we first calculate the similarity of L1 and Ln, if they are similar log records, Ln will be put into the same cluster with L1 and we go on processing next log record. If Ln and L1 are not similar, we will continue to compare the similarity between Ln and log records in other clusters until we find a suitable cluster for Ln. If Ln is not similar to any log record in all existing clusters, we will establish another new cluster for Ln. We use a running example to demonstrate the whole process in Fig. 4.

There is one key point to stress: if there're more than one log record in a cluster, we only have to choose a representative in this cluster because log records in same cluster are already similar. Thus, the complexity of the algorithm is relevant to the number of clusters which is O (n*m), in which n stands for the number of log records and m stands for the number of clusters. Taking log data set HDFS as an example (more examples can be seen in Sect. 4.1), there are 2000 log records but only 25 clusters, it's obvious that the number of log records is far more than the number of clusters ($n \gg m$), so the actual complexity of one-pass clustering algorithm is approximately equal to O(n) which is really efficient.

3.5 Merging Clusters and Identifying Variables

The results generated by one-pass clustering are not accurate enough, because the similarity measure function ***Fun1*** which calculates the ratio of identical tokens is too simple and therefore many special cases are not taken into account. As shown in Fig. 3, according to ***Fun1***, log records *blk_-5623* and *blk_-5009* will be put into different clusters because the third token is different. But it's obvious that the third token is a variable and these two log records actually belong to the same cluster, so we propose a more precise similarity measure function ***Fun2*** which reflects the ratio of similar tokens in two log records to check whether clusters need to be further merged. ***Fun2*** is defined as following:

$$Fun2(log1, log2) = \frac{\sum_{i=1}^{n} S2(log1(i), log2(i))}{\mathrm{n}} \tag{3}$$

Where *log1* and *log2* stand for two separate log records in different clusters; $log1(i)$ or $log2(i)$ represents the i-th token in this log record; n is the log length of *log1* (the length of *log1* and *log2* is the same); The function $\boldsymbol{S}2(\boldsymbol{x}, \boldsymbol{y})$ is used to determine whether two tokens are alike which is defined as following:

$$S2(x, y) = \begin{cases} 1, if\ x\ is\ similar\ to\ y \\ 0, otherwise \end{cases} \tag{4}$$

Where x and y are two tokens in log1 and log2 respectively. How to measure whether two tokens are similar is easy, we can use edit distance [16] or Cosine distance. We set another similarity threshold T* and choose a random log record from each cluster as representative. If $\boldsymbol{Fun}2(L1, L2) > \mathrm{T}^*$, we will merge the two clusters which L1 and L2 stand for. We go through all clusters in pairs and merge similar clusters, although the time complexity of merging is $O(m^2)$, this step takes up very

little time because m is a small number (m stands for the number of clusters).The following task after merging is identifying variable which is illustrated in Fig. 5. The rules for identifying is simple: *If all tokens in the same column are the same, they are constants, otherwise, they are variables.* The goal of identifying variable is telling which tokens in a log record are variables and then generating a candidate pattern for each cluster.

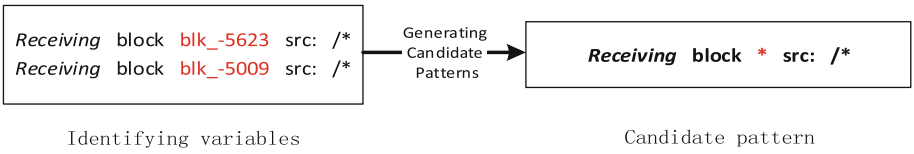

Fig. 5. Identifying variables and generating candidate patterns

3.6 Combining Candidates

We have the assumption that log records belong to the same log patterns will be very likely to share identical log length, so we group log data by length at first step. It is a smart approach because we avoid a lot of trouble caused by dealing with log records with unequal length and hence accelerate the process of clustering and merging. But nothing is perfect because log records of different log length may also belong to same log pattern, that's why we have the step of combination!

The goal of combination is to combine candidate patterns which actually come from the same log pattern. We define the instruction of combination as following: *Continuous variables will be integrated into one variable and if constant parts of two candidate log patterns are the same after they are segmented by variable, we combine them.* In order to better elaborate, we give some examples in Fig. 6.

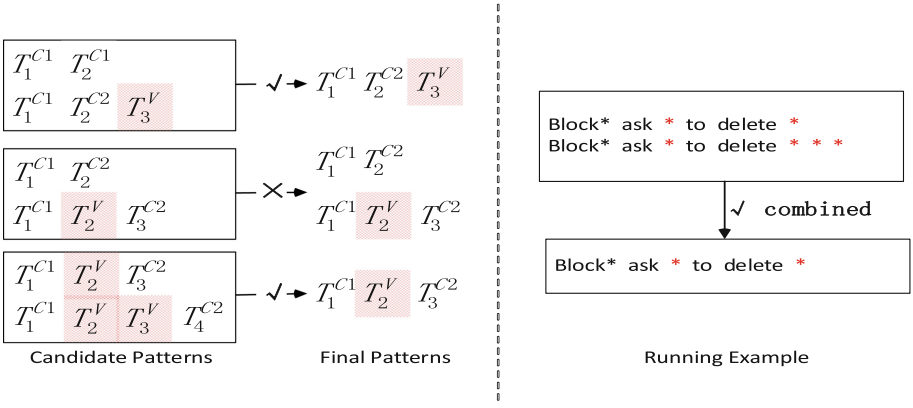

Fig. 6. Rules for combination operation

Let's explain why pattern $\{T_1^{C1}T_2^{C2}\}$ and pattern $\{T_1^{C1}T_2^{V}T_3^{C2}\}$ are not combined: There is no variable in pattern $\{T_1^{C1}T_2^{C2}\}$ so it remains as usual, but pattern $\{T_1^{C1}T_2^{V}T_3^{C2}\}$ is segmented by variable T_2^V and becomes $\{T_1^{C1}\}, \{*\}, \{T_3^{C2}\}$. It's obvious these two candidate patterns are different after segmentation so we don't combine them.

Same as the merging step, the time complexity of combination is $O(c^2)$, in which c stands for the number of candidate patterns. So time complexity of the whole algorithm is: O(n) + $O(m^2)$ + $O(c^2)$. However, the number of log records (n) is far greater than the number of clusters (m) and the number of candidate log patterns (c), taking log data set HDFS as an example (more examples can be seen in Sect. 4.1), there are 2000 log records but only 25 clusters and 16 candidate patterns, which means $n \gg m > c$. So actual time complexity of the whole algorithm is O(n).

4 Evaluation

4.1 Experimental Settings

Log Data Sets and Comparative Algorithms. The log data sets [2, 3] we used to evaluate our work are summarized in Table 1, We collect eight log data sets generated from real systems which includes distributed system logs (HDFS and Zookeeper), supercomputer logs (BGL and HPC), software logs (Proxifier) and system running logs from an anonymous Chinese financial institution (Dataset A B C).

Table 1. Summary of collected log datasets

System	Description	#Log Records	#Log Length	#Log Pattern
HDFS	Hadoop File System	11,175,629	8 to 29	116
BGL	BlueGene/L Supercomputer	4,747,963	10 to 104	376
Proxifier	Proxy Client	10,108	10 to 27	8
Zookeeper	Distributed System Coordinator	74,380	8 to 27	80
HPC	High Performance Cluster	433,490	6 to 104	105
Dataset A	System log data	3,000	5 to 17	7
Dataset B	System log data	50,314	8 to 35	78
Dataset C	System log data	126,397	8 to 33	36

We choose 4 representational existing log patterns mining methods to compare with Lopper and we test their performance based on some commonly adopted evaluation metrics. The comparative algorithms are briefly introduced as following:

- **Drain** [3]: Drain is an online log parsing tree with pre-defined rules for classifying log data and generating log patterns.
- **LogMine** [7]: LogMine is an online log pattern mining model which adopts one-pass clustering algorithm for fast log pattern mining.

- **LogSig** [10]: LogSig is an offline log pattern mining model which uses LCS (Longest Common Subsequence) for classifying log data.
- **IPLoM** [8]: IPLoM adopts a three-layer hierarchical partitioning model for clustering log records and generates log patterns from each cluster.

Table 2. Experimental environment

OS	CPU	Bits	Memory
Windows 7	Inter(R) Core(TM) i5-3230M	64	8G
Windows 7	Inter(R) Core(TM) i5-6200U	64	8G

Evaluation Metric and Experimental Setups: We use F-Measure [18] to evaluate the accuracy of Lopper, which are defined as the following:

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

Where TP stands for True Positive Rate which means true positive sample is inferred as positive. FP stands for False Positive Rate which means true negative sample is inferred as positive. FN stands for False Negative Rate which means true positive sample is inferred as negative. Experimental environment is summarized in Table 2 and we conduct each experiment several times on these two machines to reduce bias errors.

4.2 Accuracy of Lopper

Accuracy is very important because high quality log patterns are the guarantees of subsequent log mining task. We evaluate the accuracy of Lopper and other four comparative algorithms based on the log data sets described in Table 1. Because the time complexity of LogSig is O (n^2), we can't run big data set on it, therefore we randomly divide some original data sets into several log data sets with 2000 log records.

The metric "Recall" reflects the ability to mine log patterns as completely as possible. As we can see from Table 3, Lopper beats other four algorithms on 6 log data sets while IPLoM and Drain achieve the best the performance on Zookeeper (97.52%) and HPC (93.85%) respectively. LogMine has a relatively mediocre performance compared to others and LogSig is not doing very well. The metric "Precision" reflects the quality of the mined patterns which is listed at Table 4. It's obvious that Lopper still wins this game in most cases and Drain together with IPLoM follow closely.

Table 3. Recall of log pattern mining models

Log	Lopper	Drain	LogMine	IPLoM	LogSig
HDFS(2k)	**100.0%**	92.86%	87.11%	85.71%	71.43%
BGL(2K)	**96.43%**	89.29%	94.64%	95.54%	86.61%
Proxifier(2k)	**100.0%**	99.17%	98.89%	**100.0%**	87.52%
Zookeeper(2k)	96.67%	95.24%	95.24%	**97.62%**	90.48%
HPC(2k)	92.31%	**93.85%**	90.77%	95.38%	87.12%
Dataset A(3k)	**100.0%**	84.72%	80.01%	**100.0%**	71.43%
Dataset B(50k)	**91.84%**	79.15%	76.26%	86.42%	52.15%
Dataset C(126k)	**94.74%**	85.24%	84.55%	92.40%	76.25%

Table 4. Precision of log pattern mining models

Log	Lopper	Drain	LogMine	IPLoM	LogSig
HDFS(2k)	96.11%	**98.19%**	95.14%	96.25%	94.15%
BGL(2K)	83.17%	83.12%	80.72%	**84.12%**	80.14%
Proxifier(2k)	**87.17%**	86.26%	78.15%	85.20%	82.15%
Zookeeper(2k)	**93.12%**	92.15%	92.14%	90.84%	88.45%
HPC(2k)	**91.24%**	85.23%	88.96%	90.22%	60.23%
Dataset A(3k)	**85.71%**	79.43%	82.16%	83.72%	66.75%
Dataset B(50k)	**82.46%**	76.33%	79.25%	76.92%	70.21%
Dataset C(126k)	**88.89%**	77.78%	80.84%	82.08%	78.24%

Table 5. F-Measure of log pattern mining models

Log	Lopper	Drain	LogMine	IPLoM	LogSig
HDFS(2k)	**98.02%**	95.45%	90.95%	90.67%	81.23%
BGL(2K)	**89.31%**	86.09%	87.13%	89.47%	83.25%
Proxifier(2k)	**93.15%**	92.27%	87.31%	92.01%	84.75%
Zookeeper(2k)	**94.86%**	93.67%	93.66%	94.11%	89.45%
HPC(2k)	91.77%	89.33%	89.86%	**92.73%**	71.22%
Dataset A(3k)	**92.31%**	81.99%	81.07%	91.14%	69.01%
Dataset B(50k)	**86.90%**	77.71%	77.73%	81.39%	59.85%
Dataset C(126k)	**91.72%**	81.34%	82.65%	86.93%	77.23%

In order to balance "Recall" and "Precision", we use a more scientific metric "F-Measure" to evaluate the accuracy, the results are shown in Table 5. Lopper has the best performance with 92.26% on average. IPLoM is second best due to its iterative clustering structure. Drain is doing well on first five data sets but its performance has fallen sharply on Dataset A, B and C, which means Drain is not so widely applicable because Drain depends heavily on predefined rules and manual parameter adjustment.

4.3 Efficiency of Lopper

We measure the running time of Lopper and four other models to evaluate the efficiency. The experimental results are demonstrated in Table 6:

Table 6. Running time of log pattern mining models

Log(size)	Lopper	Drain	LogMine	IPLoM	LogSig
HDFS(2k)	**0.17** s	0.18 s	0.21 s	0.24 s	5.01 s
BGL(2K)	0.34 s	**0.31 s**	0.46 s	0.32 s	4.98 s
Proxifier(2k)	0.28 s	**0.26 s**	0.27 s	**0.26 s**	5.27 s
Zookeeper(2k)	**0.19 s**	0.24 s	0.21 s	0.25 s	4.90 s
HPC(2k)	**0.28 s**	0.29 s	**0.28 s**	0.31 s	4.78 s
Dataset A(3k)	0.50 s	**0.46 s**	0.68 s	0.56 s	6.53 s
Dataset B(50k)	5.17 s	**4.52 s**	8.72 s	5.40 s	137.20 s
Dataset C(126k)	**9.76** s	9.80 s	16.55 s	12.59 s	325.82 s

The time complexity of Lopper is approximately equal to O(n) which is discussed in Sect. 3.6. Drain and IPLoM share the same time complexity which is also O(n) because both of them process each log record only once. In Drain, a log record has to traverse through a tree to match the right log pattern which takes extra time. And in IPLoM, the hierarchical structure requires a lot of memory space which also impacts on execution efficiency. LogMine adopts one-pass clustering algorithm which is efficient at first, but LogMine uses Smith–Waterman Algorithm [17] which aligns two sequences of length l1 and l2 in $O(l1 * l2)$ time for merging two log records, leading to a reduction in efficiency. LogSig is the first algorithm using longest common sequence (LCS) for parsing log data, the time complexity of LCS algorithm is $O(n^2)$, so LogSig is not efficient compared to other methods.

4.4 System Anomaly Detection Task Based on Lopper

As we discussed before, log pattern mining is a preliminary work for system anomaly detection, fault diagnosis and fault root cause analysis [1, 2, 4, 6, 11]. So subsequent work after log pattern mining is the key to validating the quality of log patterns. In this section, we choose system anomaly detection task as a case study to further prove the effectiveness of Lopper.

The summarized workflow of system anomaly detection model goes like the following: Each log record corresponds to a certain log pattern, so the sequence of log records will be transferred to a sequence of log patterns. We number each log pattern so we get a sequence of numbers which stands for the original log record flow. We then set up a sliding window of fixed size to intercept log sequence and count the frequency of each log pattern in this window which will be used as feature vector. Finally, we label each window either “normal” or “abnormal” and the labels together with feature vectors are the inputs for training bi-classification models.

In this case study, we choose two commonly-used machine learning algorithms Random Forest and Decision Tree for building anomaly detection model, while Lopper and IPLoM are used for mining log patterns. The reason we choose IPLoM is that IPLoM has the best F-Measure on average among the four comparative algorithms (Drain, LogMine, IPLoM, LogSig). The results are shown in Table 7, and we can see from the experimental results that Lopper based anomaly detection model achieves an average F-Measure performance of 91.97%, which further proves the effectiveness of Lopper.

Table 7. System anomaly detection task based on Lopper and IPLoM

Lopper based anomaly detection model	Precision	Recall	F-Measure
Random Forest	100.0%	85.00%	91.89%
Decision Tree	96.15%	88.29%	92.05%
IPLoM based anomaly detection model	Precision	Recall	F-Measure
Random Forest	100.0%	81.43%	89.76%
Decision Tree	100.0%	85.50%	92.18%

5 Conclusion

System log based analysis is crucial to the management of distributed platform. To better understand system logs and depict the inner structure of log data, log pattern plays an important role. In this paper, we propose Lopper, a hybrid clustering tree for online log pattern mining. Our method accelerates the mining process by clustering raw log data in one-pass manner and ensures the accuracy by merging and combing similar patterns with different kernel functions in each step. To sum up, Lopper works in **online** mode with high **accuracy** and **efficiency**. We conduct experiments on massive sets of log data to prove the effectiveness of Lopper, the experimental results show that Lopper outperforms existing log pattern mining methods in terms of regular evaluation metrics. We also conduct aided experiments on system anomaly detection task, the results prove that log patterns generated by Lopper are quite useful for anomaly detection.

Acknowledgments. This work is supported by UINNOVA Joint Innovation project.

References

1. Lu, S., et al.: Detecting anomaly in big data system logs using convolutional neural network. In: 2018 IEEE 16th International Conference on Dependable, Autonomic and Secure Computing, 16th International Conference on Pervasive Intelligence and Computing, 4th International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech). IEEE (2018)

2. He, P., et al.: An evaluation study on log parsing and its use in log mining. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE (2016)
3. He, P., et al.: Drain: an online log parsing approach with fixed depth tree. In: 2017 IEEE International Conference on Web Services (ICWS). IEEE (2017)
4. Fu, Q., et al.: Execution anomaly detection in distributed systems through unstructured log analysis. In: 2009 Ninth IEEE International Conference on Data Mining. IEEE (2009)
5. Zhu, K.Q., Fisher, K., Walker, D.: Incremental learning of system log formats. ACM SIGOPS Operating Syst. Rev. **44**(1), 85–90 (2010)
6. Xu, W., et al.: Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. ACM (2009)
7. Hamooni, H., et al.: LogMine: fast pattern recognition for log analytics. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. ACM (2016)
8. Makanju, A.A.O., Nur Zincir-Heywood, A., Milios, E.E.: Clustering event logs using iterative partitioning. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM (2009)
9. Mizutani, M.: Incremental mining of system log format. In: 2013 IEEE International Conference on Services Computing. IEEE (2013)
10. Tang, L., Tao, L., Perng, C.-S.: LogSig: generating system events from raw textual logs. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management. ACM (2011)
11. Cheng, J., et al.: Deep convolutional neural networks for anomaly event classification on distributed systems. arXiv preprint arXiv:1710.09052 (2017)
12. Vaarandi, R.: A breadth-first algorithm for mining frequent patterns from event logs. In: Aagesen, F.A., Anutariya, C., Wuwongse, V. (eds.) INTELLCOMM 2004. LNCS, vol. 3283, pp. 293–308. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30179-0_27
13. Vaarandi, R.: A data clustering algorithm for mining patterns from event logs. In: Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No. 03EX764). IEEE (2003)
14. Du, M., Li, F.: Spell: streaming parsing of system event logs. In: 2016 IEEE 16th International Conference on Data Mining (ICDM). IEEE (2016)
15. Stearley, J.: Towards informatic analysis of syslogs. In: 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No. 04EX935). IEEE (2004)
16. Edit distance. https://en.wikipedia.org/wiki/Edit_distance
17. Smith–Waterman_algorithm. https://en.wikipedia.org/wiki/Smith-Waterman_algorithm
18. Manning, C., Raghavan, P., Schütze, H.: Introduction to information retrieval. Nat. Lang. Eng. **16**(1), 100–103 (2010)