An Evaluation Study on Log Parsing and Its Use in Log Mining

Pinjia He*†, Jieming Zhu*†, Shilin He*‡, Jian Li*† and Michael R. Lyu*†
*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong
†Shenzhen Research Institute, The Chinese University of Hong Kong, Shenzhen, China
‡School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
{pjhe, jmzhu, jianli, lyu}@cse.cuhk.edu.hk, selin.hsl@gmail.com

***Abstract*—Logs, which record runtime information of modern systems, are widely utilized by developers (and operators) in system development and maintenance. Due to the ever-increasing size of logs, data mining models are often adopted to help developers extract system behavior information. However, before feeding logs into data mining models, logs need to be parsed by a log parser because of their unstructured format. Although log parsing has been widely studied in recent years, users are still unaware of the advantages of different log parsers nor the impact of them on subsequent log mining tasks. Thus they often re-implement or even re-design a new log parser, which would be time-consuming yet redundant. To address this issue, in this paper, we study four log parsers and package them into a toolkit to allow their reuse. In addition, we obtain six insightful findings by evaluating the performance of the log parsers on five datasets with over ten million raw log messages, while their effectiveness on a real-world log mining task has been thoroughly examined.**

I. Introduction

Logs are widely used to record runtime information of software systems, such as the timestamp of an event, the unique ID of a user request, and the state of a task execution. The rich information of logs enables system developers (and operators) to monitor the runtime behaviors of their systems and further track down system problems in production settings.

With the ever-increasing scale and complexity of modern systems, the volume of logs is rapidly growing, for example, at a rate of about 50 gigabytes (around 120∼200 million lines) per hour [1]. Therefore, the traditional way of log analysis that largely relies on manual inspection has become a labor-intensive and error-prone task. To address this challenge, many efforts have recently been made to automate log analysis by the use of data mining techniques. Typical examples of log mining include anomaly detection [2], [3], [4], program verification [5], [6], problem diagnosis [7], [8], and security assurance [9], [10]. However, raw log messages are usually unstructured, because developers are allowed to record a log message using free text for convenience and flexibility. To enable automated mining of unstructured logs, the first step is to perform log parsing, whereby unstructured raw log messages can be transformed into a sequence of structured events.

Typically, a log message, as illustrated in the following example, records a specific system event with a set of fields: *timestamp* (recording the occurring time of the event), *verbosity level* (indicating the severity level of the event, e.g., INFO), and *raw message content* (recording what has happened during system operation).

```
2008-11-09 20:35:32,146 INFO dfs.DataNode$DataXceive
r: Receiving block blk_-1608999687919862906 src: /10
.251.31.5:42506 dest: /10.251.31.5:50010
```

As observed in the example, the raw message content can be divided into two parts: *constant* part and *variable* part. The constant part constitutes the fixed plain text and remains the same for every event occurrence, which can reveal the event type of the log message. The variable part carries the runtime information of interest, such as the values of states and parameters (e.g., the IP address and port: 10.251.31.5:50010), which may vary among different event occurrences. The goal of log parsing is to extract the event by automatically separating the constant part and variable part of a raw log message, and further transform each log message into a specific event (usually denoted by its constant part). In this example, the event can be denoted as "*Receiving block * src: * dest: **", where the variable part is identified and masked using asterisks. We will use "event" and "template" interchangeably in this paper.

Log parsing is essential for log mining. Traditionally, log parsing relies heavily on regular expressions to extract the specific log event (e.g., SEC [11]). However, modern software systems, with increasing size and complexity, tend to produce a huge volume of logs with diverse log events. It requires non-trivial efforts for manual creation and maintenance of regular expression rules. Especially, when a system constantly evolves, the rules of log parsing will most likely become outdated very often. For example, Google's systems, as studied in [12], have been introduced with up to thousands of new log printing statements every month. As a result, there is a high demand for automated log parsing methods, capable of evolving with the system.

To achieve this goal, recent studies have proposed a number of data-driven approaches for automated log parsing (e.g., SLCT [13], IPLoM [14], LKE [3], LogSig [15]), in which historical log messages are leveraged to train statistical models for event extraction. Despite the importance of log parsing, we found that, to date, there is a lack of systematic evaluations on

the effectiveness and efficiency of the automated log parsing methods available. Meanwhile, except SLCT [13] that was released more than 10 years ago, there are no other ready-to-use tool implementations of log parsers. Even with commercial log management solutions, such as Splunk [16] and Logstash [17], users need to provide complex configurations with customized rules to parse their logs. In this context, engineers and researchers have to implement their own log parsers when performing log mining tasks (e.g., [5], [8], [18]), which would be a time-consuming yet redundant effort. Besides, they are likely unaware of the effectiveness of their implementations compared to other competitive methods, nor do they notice the impact of log parsing on subsequent log mining tasks.

To fill this significant gap, in this paper, we perform a systematic evaluation study on the state-of-the-art log parsing methods and their employment in log mining. In particular, we intend to investigate the following three research questions:

RQ1: What is the accuracy of the state-of-the-art log parsing methods?

RQ2: How do these log parsing methods scale with the volume of logs?

RQ3: How do different log parsers affect the results of log mining?

Towards this end, we have implemented four widely-employed log parsers: SLCT [13], IPLoM [14], LKE [3], LogSig [15]. They are currently available on our Github[1] as an open-source toolkit, which can be easily re-used by practitioners and researchers for future study. For evaluation, we have also collected five large log datasets (with a total of over 10 million raw log messages) produced by production software systems. The evaluation is performed in terms of both accuracy and efficiency in log parsing. Furthermore, we evaluate the impact of different log parsers on subsequent log mining tasks, with a case study on system anomaly detection (proposed in [2]).

Through this comprehensive evaluation, we have obtained a number of insightful findings: Current log parsing methods could obtain high overall accuracy (*Finding 1*), especially when log messages are preprocessed with some domain knowledge based rules (*Finding 2*). Clustering-based log parsing methods could not scale well with the volume of logs (*Finding 3*), and the tuning of parameters (e.g., number of clusters) is time-consuming (*Finding 4*). Although highly accurate in traditional evaluation metric, current log parsing methods could not lead to optimal results of log mining (*Finding 5*), because 1% parsing errors on critical events could cause an order of magnitude performance degradation in log mining (*Finding 6*). These findings as well as our toolkit portray a picture about the current situation of log parsing methods and their effectiveness on log mining, which we believe could provide valuable guidance for future research in this field.

The remainder of this paper is organized as follows. Section II reviews the existing log parsing methods, and Section III reviews recent studies on log mining with a detailed example of anomaly detection. The evaluation results and findings are reported in Section IV. We discuss some limitations in Section V. We then introduce the related work in Section VI, and finally conclude this paper in Section VII.

[1]https://github.com/cuhk-cse/logparser

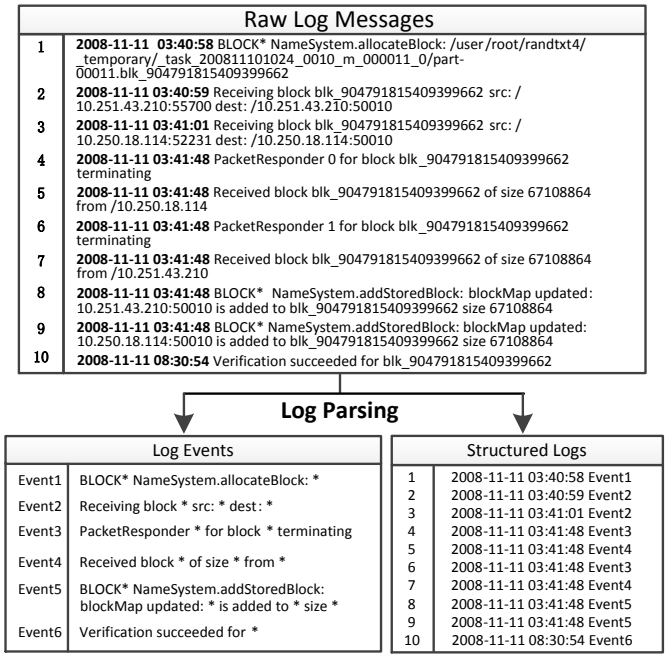


Fig. 1: Overview of Log Parsing

II. Log Parsing

This section first provides an overview of log parsing and then describes four existing log parsing methods. These methods are widely employed and thus become the main subjects of our study.

A. Overview of Log Parsing

Fig. 1 illustrates an overview of log parsing. The raw log messages, as shown in the figure, contain ten log messages extracted from HDFS log data on Amazon EC2 platform [2]. The log messages are unstructured data, with timestamps and raw message contents (some fields are omitted for simplicity of presentation). In real-world cases, a log file may contain millions of such log messages. The goal of log parsing is to distinguish between *constant* part (fixed plain text) and *variable* part (e.g., blk_ID in the figure) from the log message contents. Then, all the constant message templates can be clustered into a list of *log events*, and *structured logs* can be generated with each log message corresponding to a specific event. For instance, the log message 2 is transformed to "*Event2*" with a log template "*Receiving block * src: * dest: **". The output of a log parser involves two files with *log events* and *structured logs*. Log events record the extracted templates of log messages, while structured logs contain a sequence of events with their occurring times. Finally, the structured logs after parsing can be easily processed by log mining methods, such as anomaly detection [2] and deployment verification [6].

B. Existing Log Parsing Methods

Log parsing has been widely studied in recent years. Among all the approaches proposed, we choose four representative ones, which are in widespread use for log mining tasks. With the main focus on evaluations of these log parsing methods, we only provide brief reviews of them; the details can be found in the corresponding references.

1) SLCT

SLCT (Simple Logfile Clustering Tool) [13] is, to the best of our knowledge, the first work on automated log parsing. The work also released an open-source log parsing tool, which has been widely employed in log mining tasks, such as event log mining [19], symptom-based problem determination [20] and network alert classification [21].

Inspired by association rule mining, SLCT works as a three-step procedure with two passes over log messages: *1) Word vocabulary construction.* It makes a pass over the data and builds a vocabulary of word frequency and position. *2) Cluster candidates construction*. It makes another pass to construct cluster candidates using the word vocabulary. *3) Log template generation.* Clusters with enough log messages are selected from candidates. Then, the log messages in each cluster can be combined to generate a log template, while remaining log messages are placed into an outlier cluster.

2) IPLoM

IPLoM (Iterative Partitioning Log Mining) [22] is a log parsing method based on heuristics specially designed according to the characteristics of log messages. This method has also been used by a set of log mining studies (e.g., alert detection [4], event log analysis [23] and event summarization [24]).

Specifically, IPLoM performs log parsing through a three-step hierarchical partitioning process before template generation: *1) Partition by event size.* Log messages are partitioned into different clusters according to different lengths. *2) Partition by token position.* For each partition, words at different positions are counted. Then the position with the least number of unique words is used to split the log messages. *3) Partition by search for mapping.* Further partition is performed on clusters by searching for mapping relationships between the set of unique tokens in two token positions selected using a heuristic criterion. *4) Log template generation.* Similar to SLCT, the final step is to generate log templates from every cluster.

3) LKE

LKE (Log Key Extraction) [3] is a log parsing method developed by Microsoft, and has been applied in a set of tasks on unstructured log analysis [3], [25].

LKE utilizes both clustering algorithms and heuristic rules for log parsing: *1) Log clustering.* Raw log messages are first clustered by using hierarchical clustering algorithms with a customized weighted edit distance metric. *2) Cluster splitting.* A splitting step based on heuristic rules is performed to further split the clusters. *3) Log template generation.* The final step is to generate log templates from every cluster, similar to SLCT and IPLoM.

4) LogSig

LogSig [15] is a more recent log parsing method, which has been validated in [26].

LogSig works in three steps: *1) Word pair generation.* Each log message is converted to a set of word pairs to encode both the word and its position information. *2) Log Clustering.* Based on the word pairs, a potential value is calculated for each log message to decide which cluster the log message potentially belongs to. After a number of iterations, the log messages can be clustered. *3) Log template generation.* In each cluster, the log messages are leveraged to generate a log template.

C. Tool Implementation

Among these log parsing methods, we only found an open-source implementation on SLCT in C language. To enable our evaluations, we have implemented the other three log parsing methods in Python and also wrapped up SLCT as a Python package. For ease of use, we define standard input/output formats for these log parsers. As shown in Fig. 1, the input is a file with raw log messages, while the output contains both a file with log events and a file with structured logs. The output can be easily fed into subsequent log mining tasks. Currently, all our implementations have been open source on Github, which can be used as a toolkit for log parsing. We believe our toolkit could benefit other researchers and practitioners as well.

It is also worth noting that our current implementation targets at exactly reproducing the log parsing methods (as described in original work) for our evaluation purposes. As we will show in Section IV-C, LKE and LogSig do not scale well on large datasets. Although we plan to improve their efficiency in our future work, users may need to pay more attention when using our current toolkit.

III. Log Mining

In this section, we briefly introduce three representative log mining tasks and explain how the adopted log parsing step can affect the performance of these tasks. Further, we describe the details of a specific log mining task, system anomaly detection, which will be used for our evaluations.

A. Overview of Log Mining

Anomaly detection: Logs of Hadoop File System (HDFS) are used by Xu et al. [2] to detect anomalies in a 203-nodes HDFS. In this case, they employ source code based log parsers (not evaluated because it is beyond the scope of this paper) to find out the log events associated with each block ID, which are further interpreted with a block ID-by-event count matrix. This matrix is fed into a machine learning model to detect anomalies of the system. If the log parser adopted does not work well, some block IDs will match wrong log events, which could ruin the generated matrix and lead to failure of the anomaly detection approach.

Deployment verification: Big data application is usually developed in pseudo-cloud environment (with several PC nodes) and finally deployed in a large-scale cloud environment. Runtime analysis and debugging of such applications in deployment phase is a challenge tackled by Shang et al. in [6]. To reduce the amount of log messages which needs to be checked by developers, they compare the log event sequences generated in pseudo-cloud and large-scale cloud. Only the different log event sequences are reported to the developers, which greatly alleviates their workload. In this task, a bad log parser may produce wrong log event sequences. This could largely degrade the reduction effect because their method is based on the comparison of log event sequences.

***System model construction*:** Computer systems are difficult to debug and understand. To help developers gain insight into system behaviors, Beschastnikh et al. [5] propose a tool called Synoptic to build an accurate system model based on logs. Synoptic requires parsed log events as input and generates a finite state machine as the output system model. If an unsuitable log parser is used, both initial model building step and model refinement step will be affected. These may result in extra branches or even totally different layout of the model.

B. System Anomaly Detection

To better study the impact of log parsing approaches on the subsequent log mining task, we reproduce the anomaly detection method proposed in [2] on its original HDFS logs while using different log parsing approaches discussed in Section. II-B. The anomaly detection method contains three steps: log parsing, event matrix generation, and anomaly detection.

1) Log Parsing: The input of the anomaly detection task is a text file, each line of which is a raw log message recording an event occurring on a block in HDFS. In this step, log parsing method is adopted to figure out two things. One is all the event types appearing in the input file. The other is the events associated with each block, which distinguished by block ID. These two are exactly in the two output files of our log parser modules. We emphasize that the parsing output is not specific to anomaly detection, but also suitable for other log mining tasks.

2) Matrix Generation: Parsed results are used to generate an event count matrix Y, which will be fed into the anomaly detection model. In the event count matrix, each row represents a block, while each column indicates one event type. The value in cell $Y_{i,j}$ records how many times event j occurs on block i. We could generate Y with one pass through the parsed results. Instead of directly detecting anomaly on Y, TF-IDF [27], which is a well-established heuristic in information retrieval, is adopted to preprocess this matrix. Intuitively, TF-IDF is to give lower weights to common event types, which are less likely to contribute to the anomaly detection process.

3) Anomaly Detection: In this case, anomaly detection is to find out suspicious blocks that may indicate problems (e.g., HDFS namenode not updated after deleting a block). The model used is Principle Component Analysis (PCA) [2], which is a statistical model that captures patterns in high-dimensional data by selecting representative coordinates (principle components). PCA is used in this problem because principle components can represent most frequent patterns of events associated with blocks, which is called normal space S_d. Specifically, the first k principle components are selected to form S_d, while the remaining $n-k$ dimensions form S_a (anomaly space), where n is the number of columns (total number of event type) of the matrix. In this task, each row in the event count matrix is a vector y associated with a block. The intuition of anomaly is the vector whose end point is far away from normal space. The "distance" could be formalized by squared prediction error $SPE \equiv ||y_a||^2$, where y_a is the projection of y on S_a. y_a is calculated by $y_a = (I - PP^T)y$, where $P = [v_1, v_2, ..., v_k]$. A block is marked as anomaly if its corresponding y satisfies:

$$SPE = ||y_a||^2 > Q_\alpha,$$

where Q_α is a threshold providing $(1-\alpha)$ confidence level. For Q_α, we choose $\alpha = 0.001$ as in the original paper [2].

IV. Evaluation Study

This section presents our study methodology and reports on the detailed results for the proposed research questions.

A. Study Methodology

Log Datasets: To facilitate systematic evaluations on the state-of-the-art log parsing methods, we have used five large log datasets ranging from supercomputers (BGL and HPC) to distributed systems (HDFS and Zookeeper) to standalone software (Proxifier), with a total of 16,441,570 lines of log messages. Table I provides a basic summarization of these datasets. Logs are scarce data for research, because companies are often reluctant to release their production logs due to confidentiality issue. We obtained three log datasets, with the generous support from their authors. Specifically, BGL is an open dataset of logs collected from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL), with 131,072 processors and 32,768GB memory [28]. HPC is also an open dataset with logs collected from a high performance cluster at Los Alamos National Laboratory, which has 49 nodes with 6,152 cores and 128GB memory per node [29]. HDFS logs are collected in [2] by using a 203-node cluster on Amazon EC2 platform. To enrich the log data for evaluation purpose, we further collected two datasets: one from a desktop software Proxifier, and the other from a Zookeeper installation on a 32-node cluster in our lab.

In particular, the HDFS logs from [2] have well-established anomaly labels, each of which indicates whether or not a request for a data block operation is an anomaly. The labels are made based on domain knowledge, which are suitable for our evaluations on anomaly detection with different log parsers. Specifically, the dataset with over 11 million log messages records 575,061 operation requests with a total of 29 event types. Among all the 575,061 requests, 16,838 are marked as anomalies, which we use as ground truth in our evaluation.

TABLE I: Summary of Our System Log Datasets

System	Description	#Logs	Length	#Events
BGL	BlueGene/L Supercomputer	4,747,963	10∼102	376
HPC	High Performance Cluster (Los Alamos)	433,490	6∼104	105
Proxifier	Proxy Client	10,108	10∼27	8
HDFS	Hadoop File System	11,175,629	8∼29	29
Zookeeper	Distributed System Coordinator	74,380	8∼27	80

TABLE II: Parsing Accuracy of Log Parsing Methods (Raw/Preprocessed)

	BGL	HPC	HDFS	Zookeeer	Proxifier
SLCT	**0.61/0.94**	0.81/0.86	0.86/0.93	0.92/0.92	0.89/-
IPLoM	0.99/0.99	0.64/0.64	0.99/1.00	0.94/0.90	0.90/-
LKE	0.67/0.70	0.17/0.17	**0.57/0.96**	0.78/0.82	0.81/-
LogSig	**0.26/0.98**	0.77/0.87	0.91/0.93	0.96/0.99	0.84/-

***Experimental Setup*:** All our experiments were run on a Linux server with Intel Xeon E5-2670v2 CPU and 128GB DDR3 1600 RAM, running 64-bit Ubuntu 14.04.2 with Linux kernel 3.16.0. We use F-measure [30], [31], a commonly-used evaluation metric for clustering algorithms, to evaluate the parsing accuracy of log parsing methods. To calculate F-measure, we manually obtain the ground truths for all logs of these dataset. It is possible because we iteratively filter out logs with confirmed event using regular expression. Experiments about LKE and LogSig are run 10 times to avoid bias of clustering algorithms, while others are run once because they are deterministic. We note here that only the parts of free-text log message contents are used in evaluating the log parsing methods.

B. RQ1: Accuracy of Log Parsing Methods

To study the accuracy of different log parsing methods, we use them to parse our collected real logs. As with the existing work [15], we randomly sample 2k log messages from each dataset in our evaluation, because the running time of LKE and LogSig is too long on large log datasets (e.g., LogSig requies 1 day to parse entire BGL data). The average results of 10 runs are reported in Table II. We can observe that the overall accuracy of these log parsing methods is high (larger than 0.8 in most cases). Meanwhile, the overall accuracy on HDFS, Zookeeper and Proxifier datasets is higher than that obtained on the others. We found that this is mainly because BGL and HPC logs involve much more event types, each of which has a longer length than other datasets.

Especially, we found that LKE takes an aggressive clustering strategy, which groups two clusters if any two log messages between them has a distance smaller than a specified threshold. This is why LKE has an accuracy drop on HPC dataset, in which it clusters almost all the log messages into one single cluster in the first step. BGL contains a lot of log messages whose event is "*generating core.**", such as "generating core.2275" and "generating core.852". Intuitively, the similarity of these two log messages are 50%, because half of the words are different. LogSig tends to separate these log messages into different clusters, which causes its low accuracy on BGL. Particularly, IPLoM leverages some heuristic rules developed on the characteristics of log messages, while other log parsing methods rely on well-studied data mining models. However, we found that IPLoM obtains the superior overall accuracy (0.88) against other log parsing methods. This further implies the particular importance of exploiting the unique characteristics of log data in log parsing, which would shed light on future design and improvement of a log parser.

> *Finding 1*: Current log parsing methods achieve high overall parsing accuracy (F-measure).

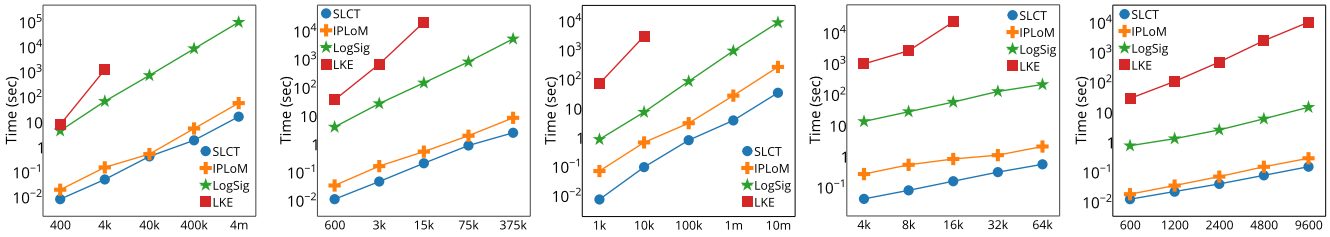
Instead of running log parsing methods directly on raw log messages, developers usually preprocess log data with domain knowledge. In this experiment, we study the impact of preprocessing on parsing accuracy. Specifically, we remove obvious numerical parameters in log messages (i.e., IP addresses in HPC&Zookeeper&HDFS, core IDs in BGL, and block IDs in HDFS). Proxifier does not contain words that could be preprocessed based on domain knowledge. Preprocessing is mentioned in LKE and LogSig; however, its importance has not been studied.

In Table II, the numbers on the left/right side represent the accuracy of log parsing methods on raw/preprocessed log data. In most cases, accuracy of parsing is improved. Preprocessing greatly increases the accuracy of SLCT/LKE/LogSig on one dataset (in bold). However, preprocessing could not improve the accuracy of IPLoM. It even slightly reduces IPLoM's accuracy on Zookeeper. This is mainly because IPLoM considers preprocessing internally in its four-step process. Unnecessary preprocessing may cause wrong splitting.

> *Finding 2:* Simple log preprocessing using domain knowledge (e.g. removal of IP address) can further improve log parsing accuracy.

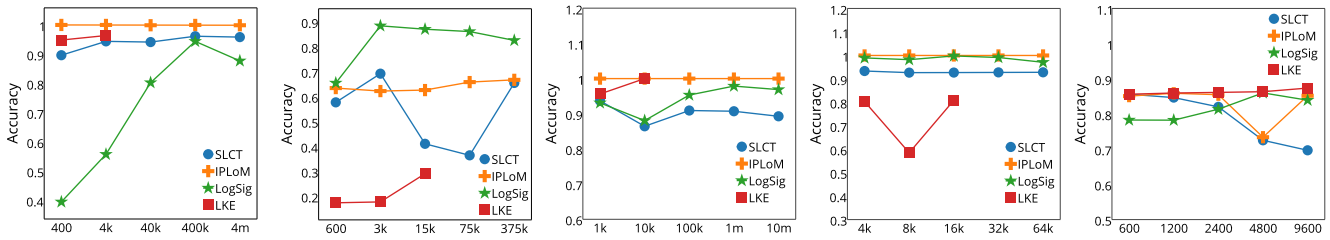
C. RQ2: Efficiency of Log Parsing Methods

In Fig. 2, we evaluate the running time of the log parsing methods on all datasets by varying the number of raw log messages. Notice that as the number of raw log messages increases, the number of events becomes larger as well (e.g., 60 events in BGL400 while 206 events in BGL40k). SLCT and IPLoM, which are based on heuristic rules, scale linearly with the number of log messages (note that Fig. 2 is in logarithmic scale). Both of them could parse 10 million HDFS log messages within five minutes. For the other two clustering-based parsing methods, LogSig also scales linearly with the number of log messages. However, its running time also increases linearly with the number of events, which leads to relatively longer parsing time (e.g, 2+ hours for 10m HDFS log messages). The time complexity of LKE is $O(n^2)$, which makes it unable to handle large-scale log data, such as BGL4m and HDFS10m. Some running time of LKE is not plotted because LKE could not parse some scales in a reasonable



(a) BGL (b) HPC (c) HDFS (d) Zookeeper (e) Proxifier

Fig. 2: Running Time of Log Parsing Methods on Datasets in Different Size



(a) BGL (b) HPC (c) HDFS (d) Zookeeper (e) Proxifier

Fig. 3: Parsing Accuracy on Datasets in Different Size

time (may cause days or even weeks). To reduce the running time of clustering-based log parsing method, parallelization is a promising direction.

Finding 3: Clustering-based log parsing methods could not scale well on large log data, which implies the demand for parallelization.

The accuracy of log parser is affected by parameters. For example, the number of clusters of LogSig decides the number of events, which should be set beforehand. For large-scale log data, it is difficult to select the most suitable parameters by trying different values, because each run will cause a lot of time. A normal solution is to tune the parameters in a sample dataset and directly apply them on large-scale data. To evaluate the feasibility of this approach, we tune parameters for log parsing methods on 2k sample log messages, which are used in our parsing accuracy experiment. In Fig. 3, we vary the size of the dataset and evaluate the accuracy of the log parsing method using these parameters. The results show that the IPLoM performs consistently in most cases. SLCT is also consistent in most cases except HPC. The accuracy of LKE is volatile because of the weakness of its clustering algorithm discussed in Section IV-B. LogSig performs consistently on datasets with limited types of events, but its accuracy varies a lot on datasets with many events (i.e., BGL and HPC). Thus, for LKE and LogSig, directly using parameters tuned on sample dataset is not practical, which makes parameter tuning on large-scale logs time-consuming.

Finding 4: Parameter tuning for clustering-based log parsing methods is a time-consuming task, especially on large log datasets.

TABLE III: Anomaly Detection with Different Log Parsing Methods

	Parsing Accuracy	True Anomaly	Detected Anomaly	False Alarm
SLCT	0.90	16,838	391 (2.3%)	5,366 (93%)
LogSig	0.85	16,838	704 (4.2%)	12,589 (95%)
IPLoM	0.99	16,838	13,874 (82%)	21,268 (60%)
IPLoM+	1.00	16,838	11,473 (68%)	278 (2.3%)
Ground truth	1.00	16,838	11,473 (68%)	278 (2.3%)

D. RQ3: Effectiveness of Log Parsing Methods on Log Mining

To evaluate the effectiveness of log parsing methods on log mining, we use three log parsers to tackle the parsing challenge of a real-world anomaly detection task described in Section III-B. The parameters of them are re-tuned to provide good Parsing Accuracy. LKE is not employed because it could not handle this large amount of data (10m+ lines) in reasonable time. IPLoM+ enhanced IPLoM with postprocessing, which is discussed in detail in *Finding 6*. The evaluation results are illustrated in Table III. True Anomaly means the number of anomalies found manually in [2]. Detected Anomaly is the number of true anomalies detected by PCA, while adopting different log parsers in log parsing step. False Alarm means the number of wrongly detected anomalies. Ground truth is the experiment using exactly correct parsed results in anomaly detection. Notice that even the Ground truth could not detect all anomalies because of the boundary of the PCA anomaly detection model.

From Table III, we observe that the parsing accuracy of these parsing methods are high. However, the false alarm rate of them is an order of magnitude lower than ground truth. This is also because some log mining algorithms such as PCA is

very sensitive to the correctness of input data. In any case, the result still reveals that these log parsing methods are not good enough to tackle real-world log mining problems.

Finding 5: Although highly accurate in traditional evaluation metric (namely F-measure), current log parsing methods could not lead to optimal results on log mining tasks.

To further analyze the effectiveness of log parsing methods on log mining, we manually verify the correctness of all the log messages parsed by IPLoM beacuse its parsing accuracy is the highest in Table III. We observe that only 7,891 log messages out of 11,175,629 are wrongly parsed, which causes 1% parsing accuracy loss. To evaluate the impact of this tiny parsing error, we manually match these 7,891 log messages to the correct event and feed the manually postprocessed log messages to the anomaly detection model. Then the model (IPLoM+) performs exactly the same as the Ground truth in Table III. This shows that the 1% error in parsing could cause the false alarm rate raising from 2.3% to 60% in anomaly detection task, which brings about extensive manual verification work. It also implies that an evaluation metric, which can reflect true effectiveness of a log parsing method on subsequent log mining tasks, is in demand.

Finding 6: Due to the impact of some critical events, 1% error in parsing could even cause an order of magnitude performance degradation in log mining.

V. Discussions

Limitations: **1) Diversity of dataset.** Not all datasets (two out of five) used in our evaluation are production data. This is mainly because of the lack of public log data. We thanks those who release log data [2], [29], [28], which greatly facilitates our research. However, Zookeeper and HDFS are popular systems adopted by many companies for their distributed computing jobs. We believe these logs could reflect the logs from industrial companies to some extent. **2) Diversity of log mining tasks.** Results of effectiveness of log parsing methods are evaluated on anomaly detection, which may not generalize to other log mining tasks. This mainly because log mining task with released real-world data is scarce. However, the anomaly detection task evaluated is presented in a paper [2] with more than 250 citations, which is an important log mining task widely studied [32], [33]. Besides, even conducting evaluation on one log event mining task, our result reveal the inconspicuous fact that current log parsing methods with high accuracy could not lead to optimal result of log mining. We will consider to extend our methodology on more varied log data as well as log mining tasks in our future work.

Potential Directions: **1) Distributed Log Parsing.** Our experiments show that current log parsing methods cost a lot of time on big data input. The amount of log message in industrial companies could be much larger. Log parsing methods based on heuristic rules are fast but their parsing result is not good enough to fulfill the need of log mining task. Thus, to accelerate the parsing process and further improve its accuracy, log parsing methods which run in a distributed manner are in demand. Clustering algorithms which could be parallelized should be considered. **2) Logging of Event ID.** We could also improve log parsing process by recording event ID in logs in the first place. This approach is feasible because developer writing log knows exactly which event a log message statement match. Thus, adding event ID to log message is a good logging practice [34] from the perspective of log mining. Tools that could automatically add event ID into source code may greatly facilitate the log parsing process.

VI. Related Work

***Log Analysis*:** Logs, as an important data source, are in widespread use for system management tasks, such as anomaly detection [3], [2], program verification [5], [6], performance monitoring [8], [7], security assurance [9], [10], failure analysis [35], etc. As shown in our evaluation results, log parsing is a critical step to enable effective log analysis. Thus, we believe our work on log parsing could benefit future studies on log analysis.

***Log Parsing*:** Log parsing has been widely studied. Xu et al. [2] implement a log parser with very high accuracy based on source code analysis to infer log message templates. However, in practice, source code is often unavailable or incomplete to access, especially when third-party components are employed. Some other work proposes data-driven approaches to log parsing (e.g., SLCT [13], IPLoM [22], LKE [3], LogSig [15]), in which data mining techniques are leveraged to extract log message templates. But there is currently a lack of open-source implementations on log parsing tools. Many researchers (e.g., [5], [8], [18]) and practitioners (as revealed in StackOverflow questions [36], [37]) in this field have to implement their own log parsers to deal with their log data. This is a time-consuming yet redundant task. Our work not only provides valuable insights on log parsing, but also releases open-source tool implementations on the state-of-the-art log parsing methods.

***Empirical Study*:** Empirical studies have attracted considerable attraction in recent years, because the empirical results could usually provide useful insights and direct suggestions to both academic researchers and industrial practitioners. In particular, Yuan et al. [38], [7] perform a characteristic study on the logging practices in open-source systems and further provide actionable suggestions for improvement. Meanwhile, some recent work [39], [40], [41] has studied the logging practices in industry. Our work is another empirical study, with a focus on evaluations on log parsing and its use in log mining.

VII. Conclusion

Log parsing is employed pervasively in log mining. However, due to the lack of studies on performance of log parsing methods, users often re-design a specialized log parser, which is time-consuming. In this paper, we study the performance of four state-of-the-art log parsing methods through extensive experiments. We also analyze the effectiveness of the log

parsing methods on a real-world log mining task with 10 million log messages. We provide six valuable findings on the parsing accuracy of the log parsers, efficiency of the log parsers, and their effectiveness on log mining. In addition, the source code of these log parsing methods is released for reuse and further study.

ACKNOWLEDGMENT

The work described in this paper was fully supported by the National Natural Science Foundation of China (Project No. 61332010), the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14205214 of the General Research Fund), and 2015 Microsoft Research Asia Collaborative Research Program (Project No. FY16-RES-THEME-005).

REFERENCES

[1] H. Mi, H. Wang, Y. Zhou, R. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, pp. 1245–1255, 2013.

[2] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordon, "Detecting large-scale system problems by mining console logs," in *SOSP'09: Proc. of the ACM Symposium on Operating Systems Principles*, 2009.

[3] Q. Fu, J. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *ICDM'09: Proc. of International Conference on Data Mining*, 2009.

[4] A. Makanju, A. Zincir-Heywood, and E. Milios, "Fast entropy based alert detection in super computer logs," in *DSN-W'10: Proc. of International Conference on Dependable Systems and Networks Workshops*, 2010, pp. 52–58.

[5] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. Ernst, "Leveraging existing instrumentation to automatically infer invariant-constrained models," in *ESEC/FSE'11: Proc. of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011.

[6] W. Shang, Z. Jiang, H. Hemmati, B. Adams, A. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *ICSE'13: Proc. of the 35th International Conference on Software Engineering*, 2013, pp. 402–411.

[7] D. Yuan, S. Park, P. Huang, Y. Liu, M. Lee, X. Tang, Y. Zhou, and S. Savage, "Be conservative: enhancing failure diagnosis with proactive logging," in *OSDI'12: Proc. of the 10th USENIX Conference on Operating Systems Design and Implementation*, 2012, pp. 293–306.

[8] K. Nagaraj, C. Killian, and J. Neville, "structured comparative analysis of systems logs to diagnose performance problems," in *NSDI'12: Proc. of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.

[9] A. Oprea, Z. Li, T. Yen, S. Chin, and S. Alrwais, "Dectection of early-stage enterprise infection by mining large-scale log data," in *DSN'15*, 2015.

[10] Z. Gu, K. Pei, Q. Wang, L. Si, X. Zhang, and D. Xu, "Leaps: Detecting camouflaged attacks with statistical learning guided by program analysis," in *DSN'15*, 2015.

[11] D. Lang, "Using SEC," *USENIX ;login: Magazine*, vol. 38, no. 6, pp. 38–43, 2013.

[12] W. XU, "System problem detection by mining console logs," Ph.D. dissertation, University of California, Berkeley, 2010.

[13] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IPOM'03: Proc. of the 3rd Workshop on IP Operations and Management*, 2003.

[14] A. Makanju, A. Zincir-Heywood, and E. Milios, "Clustering event logs using iterative partitioning," in *KDD'09: Proc. of International Conference on Knowledge Discovery and Data Mining*, 2009.

[15] L. Tang, T. Li, and C. Perng, "LogSig: generating system events from raw textual logs," in *CIKM'11: Proc. of ACM International Conference on Information and Knowledge Management*, 2011, pp. 785–794.

[16] Splunk. [Online]. Available: http://www.splunk.com

[17] Logstash. [Online]. Available: http://logstash.net

[18] S. Banerjee, H. Srikanth, and B. Cukic, "Log-based reliability analysis of software as a service (saas)," in *ISSRE'10: Proc. of the 21st International Symposium on Software Reliability Engineering*, 2010.

[19] R. Vaarandi, "Mining event logs with slct and loghound," in *NOMS'08: Proc. of the IEEE/IFIP Network Operations and Management Symposium*, 2008.

[20] L. Huang, X. Ke, K. Wong, and S. Mankovskii, "Symptom-based problem determination using log data abstraction," in *CASCON'10 Proc. of the Conference of the Center for Advanced Studies on Collaborative Research*, 2010, pp. 313–326.

[21] R. Vaarandi and K. Podis, "Network ids alert classification with frequent itemset mining and data clustering," in *CNSM'10: Proc. of the Conference on Network and Service Management*, 2010.

[22] A. Makanju, A. Zincir-Heywood, and E. Milios, "A lightweight algorithm for message type extraction in system application logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, pp. 1921–1936, 2012.

[23] ——, "Investigating event log analysis with minimum apriori information," in *IM'13: Prof. of International Symposium on Integrated Network Management*, 2013, pp. 962–968.

[24] Y. Jiang, C. Perng, and T. Li, "Meta: Multi-resolution framework for event summarization," in *SDM'14: Proc. of the SIAM International Conference on Data Mining*, 2014, pp. 605–613.

[25] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *ATC'10: Proc. of the USENIX Annual Technical Conference*, 2010.

[26] L. Tang, T. Li, L. Shang, F. Pinel, and G. Grabarnik, "An integrated framework for optimizing automatic monitoring systems in large it infrastructures," in *KDD'13: Proc. of International Conference on Knowledge Discovery and Data Mining*, 2013, pp. 1249–1257.

[27] G. Salton and C. Buckley, "Term weighting approaches in automatic text retrival," Cornell, Tech. Rep., 1987.

[28] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *DSN'07*, 2007.

[29] L. A. N. S. LLC. Operational data to support and enable computer science research. [Online]. Available: http://institutes.lanl.gov/data/fdata

[30] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[31] Evaluation of clustering. [Online]. Available: http://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html

[32] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *SIGCOMM'04: Proc. of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2004, pp. 219–230.

[33] H. Ringberg, A. Soule, J. Rexford, and C. Diot, "Sensitivity of pca for traffic anomaly detection," in *SIGMETRICS'07: Proc. of International Conference on Measurement and Modeling of Computer Systems*, 2007.

[34] F. Salfner, S. Tschirpke, and M. Malek, "Comprehensive logfiles for autonomic systems," in *IPDPS'04: Proc. of the 18th International Parallel and Distributed Processing Symposium*, 2004.

[35] C. Di Martino, M. Cinque, and D. Cotroneo, "Assessing time coalescence techniques for the analysis of supercomputer logs," in *DSN'12*, 2012.

[36] [Online]. Available: http://stackoverflow.com/questions/154982/what-is-the-best-log-analysis-tool-that-you-used

[37] [Online]. Available: http://stackoverflow.com/questions/2590251/is-there-a-log-file-analyzer-for-log4j-files

[38] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in *ICSE'12: Proc. of the 34th International Conference on Software Engineering*, 2012, pp. 102–112.

[39] Q. Fu, J. Zhu, W. Hu, J. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," in *ICSE'14: Companion Proc. of the 36th International Conference on Software Engineering*, 2014, pp. 24–33.

[40] J. Zhu, P. He, Q. Fu, H. Zhang, R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *ICSE'15: Proc. of the 37th International Conference on Software Engineering*, 2015.

[41] A. Pecchia, M. Cinque, G. Carrozza, and D. Cotroneo, "Industry practices and event logging: assessment of a critical software development process," in *ICSE'15: Proc. of the 37th International Conference on Software Engineering*, 2015, pp. 169–178.