Revisiting Log Parsing: The Present, The Future, and The Uncertainties

Zhijing Li, Qiuai Fu, Zhijun Huang, Jianbo Yu, Yiqian Li, Yuanhao Lai, Yuchi Ma, Pinjia He

***Abstract*—In the recent decade, the amount of software runtime logs has increased rapidly and spawned a line of automated log analysis research using machine learning or data mining algorithms. In the typical workflow of log analysis, log parsing, which aims to transform unstructured or semi-structured logs into structured logs, is crucial to various downstream algorithms. While the state-of-the-art (SOTA) parsers achieve extremely high accuracy, recent research shows that these parsers are far from being useful under stricter evaluation metrics. Thus, researchers and practitioners are unclear about the current state of log parsing research and what might be important to explore in the future. To this end, we conduct an empirical study to revisit log parsing by running extensive experiments of SOTA parsers on sixteen widely-used log datasets under five evaluation metrics with different preprocessing settings. Our results show that the performance of log parsers varies significantly under different evaluation metrics. In addition, preprocessing plays an important role in the evaluation. In particular, preprocessing with common regular expressions can cause a 0.38 performance difference in group accuracy (GA), highlighting the importance of reporting preprocessing details in parsing research. We also generalize the word-level regular expressions in preprocessing and try to use them to parse the whole logs, which leads to surprisingly decent accuracy. These results imply that formulating log parsing as a word-level classification task is a feasible future direction. Moreover, we find out that the most widely-used dataset (i.e., LogHub) is suboptimal because it contains labeling errors. To address this issue, we make an extensive manual effort to fix the errors in the log dataset, providing a revised Ground Truth for future log parsing research. On the revised log dataset, our simple parser (word-level regex-based) achieves 0.97 Precision-template accuracy (PTA) on the Spark dataset and an average Recall-template accuracy (RTA) of 0.93 on 16 datasets, which outperforms all existing parsers. We gathered log data from actual companies and evaluated all parsers. Our regex-based parser achieved a PTA value of 0.9565, surpassing other parsers by 0.7642.**

***Index Terms*—Log parsing, Log analysis, Empirical study, Dataset**

I. Introduction

SOFTWARE logs are a chronological collection of specific system operations and their operation outcomes. Logs are generated by logging statements written by developers in the source code, and thus, logs often contain structured information (e.g., timestamp) and unstructured natural language text. As the system generates much more logs, managing and analyzing them becomes increasingly challenging. To address this problem, recent research has focused on designing automated log management and analysis approaches.

Log parsing is an essential aspect of system reliability engineering. It helps practitioners identify and troubleshoot system failures by analyzing log files generated by software systems. These log files contain a wealth of information that can provide insights into system behavior, performance, and errors. Previous research has highlighted that minor parsing errors can have a significant impact on the efficacy of downstream processes [1]. Furthermore, IBM utilizes log analysis for monitoring network outages [1]. Therefore, accurate and efficient log parsing is critical for ensuring system reliability. In this context, the study of log parsing methods, their accuracy, and efficiency is an important area of research within the broader field of reliability engineering. The use of log parsing tools and techniques can help reduce the time and effort required to diagnose system issues, thereby improving system reliability. Furthermore, as software systems become increasingly complex and larger volumes of data are generated, log parsing is becoming even more critical for ensuring system reliability. In summary, log parsing is an integral part of reliability engineering, and the study of log parsing accuracy and efficiency is crucial for improving system reliability. The paper's contribution to the field lies in evaluating the performance of various log parsing methods and providing insights into how these methods can be improved to enhance system reliability.

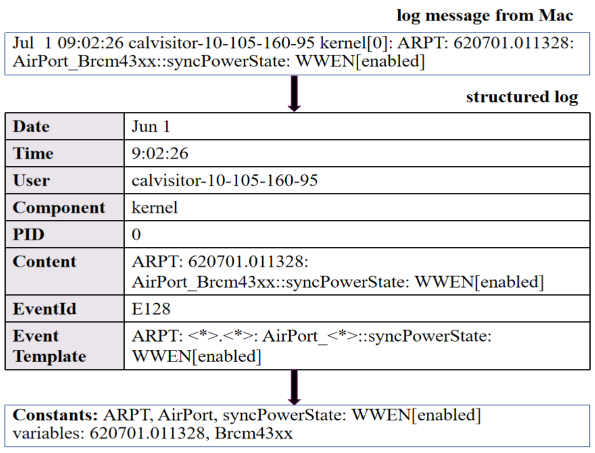
In the workflow of log management and analysis, log parsing, which transforms unstructured or semi-structured logs into structured logs, is the first and foremost step. Typically, a log message contains constants, which are tokens written by developers in logging statements (e.g., a description of system operation), and variables, which exhibit different values during runtime (e.g., timestamps and program states). A constant, which is the same for each occurrence of the event, displays the event template for the log message, while a variable includes dynamic information that can change depending on the circumstances of certain events. The output of a log parser is a log event template and a list of structured log messages. Fig. 1 shows an example of log parsing. Date, Time, User, Component, PID, Content, EventId, and Event Template are all included in the parsed log message from the Mac dataset. The variables are converted into user-defined symbols (e.g., ARPT:

Manuscript created February, 2023; This work was developed by the IEEE Publication Technology Department. Corresponding author: Pinjia He. Zhijing Li, Zhijun Huang, Jianbo Yu, Yiqian Li, and Pinjia He are with the School of Data Science, Chinese University of HongKong, Shenzhen.(E-mail: zhijingbaby@gmail.com, {119020019, jianboyu, 120090702}@link.cuhk.edu.cn, hepinjia@cuhk.edu.cn) Qiuai Fu and Yuchi Ma are with HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.(E-mail: fuqiuai@huawei.com, mayuchi1@huawei.com) Yuanhao Lai is with the SERVICE LAB, HUAWEI TECHNOLOGIES.(E-mail:laiyuanhao@huawei.co)

[1]https://developer.ibm.com/blogs/how-mining-log-templates-can-help-ai-ops-in-cloud-scale-data-centers/

TABLE I
SUMMARY OF LOG DATASETS.

Dataset	Time Span	Data Size	Messages	Templates (total)	Templates (2k)
HDFS	38.7 hours	1.47 GB	11,175,629	30	14
Hadoop	N.A	48.61 MB	394,308	298	114
Spark	N.A	2.75 GB	33,236,604	456	36
ZooKeeper	26.7 days	9.95 MB	74,380	95	50
OpenStack	N.A.	60.01 MB	207,820	51	43
BGL	214.7 days	708.76 MB	4,747,963	619	120
HPC	N.A.	32.00 MB	433,489	104	46
Thunderbird	244 days	29.60 GB	211,212,192	4040	149
Windows	226.7 days	26.09 GB	114,608,388	4833	50
Linux	263.9 days	2.25 MB	25,567	488	118
Mac	7.0 days	16.09 MB	117,283	2,214	341
Android	N.A.	3.38 GB	30,348,042	76,923	166
HealthApp	10.5 days	22.44 MB	253,395	220	75
Apache	263.9 days	4.90 MB	56,481	44	6
OpenSSH	28.4 days	70.02 MB	655,146	62	27
Proxifier	N.A.	2.42 MB	21,329	9	8


log message from Mac

```
Jul  1 09:02:26 calvisitor-10-105-160-95 kernel[0]: ARPT: 620701.011328:
AirPort_Brcm43xx::syncPowerState: WWEN[enabled]
```

structured log

Date	Jun 1
Time	9:02:26
User	calvisitor-10-105-160-95
Component	kernel
PID	0
Content	ARPT: 620701.011328: AirPort_Brcm43xx::syncPowerState: WWEN[enabled]
EventId	E128
Event Template	ARPT: <*>.<*>: AirPort_<*>::syncPowerState: WWEN[enabled]

Constants: ARPT, AirPort, syncPowerState: WWEN[enabled]
variables: 620701.011328, Brcm43xx

Fig. 1. An example of log parsing.

620701.011328 is converted into ARPT: <*>). The benefit of log parsing is that the structured logs and templates can be easily adopted by subsequent tasks such as log processing, analysis, and querying, which largely facilitates the whole workflow.

At present, there are two main trends in research for log parsing: (1) design a novel log parser that achieves SOTA accuracy, such as Drain [2], MoLFI [3], and LogClass [4]; (2) propose a new evaluation metric and claim that the current one is not good enough (such as the paper proposed by khan et al. [5], Logram [6] and Uniparser [7]). Zhu et al. performed a thorough examination of automated log parsers and then published the results as reusable benchmarks [8]. More specifically, the authors tested thirteen log parsers against sixteen log datasets from distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software. Khan et al.[5] assessed log parsers in terms of metrics based on this research. The authors believed that these metrics have a few flaws, Khan et al. evaluated the properties of existing metrics and developed their own (PTA and RTA) for a more thorough evaluation of log parsers. The outcomes of all parsers considerably declined in these rigorous metrics testing. Thus, the authors indicated that log parsers are still far from perfect and there are a lot of opportunities for further development. Moreover, the authors discussed possible correlations between various metrics. In addition, the authors mention that the dataset (i.e., event templates and structured logs) might contain labeling errors (e.g., a wrong event template).

These existing papers have largely facilitated the development of log analysis research. However, some important issues have not been well explored or addressed. First, to the best of our knowledge, none of the existing papers systematically explored the impact of preprocessing on parsing accuracy, which we argue is important. Second, most of the papers [9] in this field formalize log parsing as a clustering problem that clusters log messages with the same event template into the same group (log message-wise). Meanwhile, a few recent papers [10] start to regard log parsing as a classification problem that classifies each token in a log message as either a *constant* or a *variable* (token-wise). While log message-wise approaches outperform manual approaches that construct regular expressions (regex) for every possible log message, it is unclear whether regex can be extensively used in token-wise approaches. Third, none of the existing papers systematically fix the labeling errors in LogHub [11], the most widely-adopted dataset in this field, which could lead to the unreliability of the results reported by many log parsing papers [7].

To fill these significant gaps, in this work, we conduct an empirical study to revisit log parsing by running extensive experiments of thirteen well-known parsers on sixteen widely-used datasets under five evaluation metrics. The results reveal that preprocessing has a significant impact on parsing accuracy (e.g., 0.38 performance difference in group accuracy), highlighting the importance of reporting preprocessing details in this field. In addition, we construct a token-wise toy parser that consists of only 23 simple token-level regexes, which achieve surprisingly decent performance on most of the datasets. We believe these results demonstrate the potential of the idea that regards log parsing as a classification problem and leverage token-level regexes. The parsing results have been greatly improved under all metrics, especially PTA and RTA. Moreover, we invested a huge amount of manual effort (21 man-months) to manually revisit the labels in LogHub datasets and spotted 587 templates errors that affect over 14,200 log messages. The revised datasets have been merged with LogHub in its newest release.

The evaluation metrics and log parsing techniques proposed in the paper can help improve the reliability of systems that generate large amounts of log data in several ways. First, these techniques can help identify and diagnose errors, warnings, and other issues in log data more accurately and quickly, allowing system administrators and developers to address them before they become more serious problems. Second, by improving the accuracy and efficiency of log parsing, these techniques can help reduce the amount of time and effort required to analyze log data, enabling faster and more effective troubleshooting and maintenance of systems. Finally, by enabling better log data analysis and management, these techniques can help improve overall system performance and reliability by identifying patterns and trends in log data that may indicate emerging issues or areas for optimization. Overall, the use of effective log parsing techniques and evaluation metrics can help organizations better understand and improve the reliability of their systems, leading to increased uptime, reduced downtime, and improved user satisfaction. The main contributions of this paper are as follows:

- It is the first empirical study that systematically explores the impact of preprocessing in log parsing.
- It studies a simple log parser that consists of 23 token-level regexes, which outperforms the current popular parsers in all metrics, demonstrating the potential of token-level parsers with regexes.
- It provides a brand-new version of sixteen widely-used log datasets with extensive manual effort, which provide a better benchmark for log parsing research.[2]

The paper has the following structure: Section 1 (Introduction) explains the importance of log parsing, the problems it solves, and the scope and main contributions of the paper. Section 2 (Related work) presents the log parsers to be evaluated in the article, including their methods and characteristics. Section 3 (Present) presents the experimental data, the effect of preprocessing on log parsing, evaluation metrics, and the results of our experiments on parsing errors. Section 4 (Future) describes how we modified the original ground truth and designed a regex-based parser, followed by the results of our experiments to verify their effectiveness. Section 5 (Uncertainties) discusses uncertainties in the modification process and experiments. Section 6 (Threats) identifies open issues for log parsing technology. Finally, Section 7 (Conclusions) concludes the paper and suggests future work directions.

[2]https://github.com/logpai/logparser/tree/master/logsrevised

II. Related Work

In this section, we talk about different ways to parse logs, such as fixed time windows, sliding time windows, session windows, and other methods. Log parsing is important for log analysis and it can find related event templates from unstructured logs.

A. Log parsing

Log parsing is the process of extracting related event templates from unstructured logs, each of which is made up of many parameters and serves as the foundation for feature extraction. Fixed time windows, sliding time windows, session windows, and other log segmentation methods are now employed due to the exploding number of logs. The detection of these logs is conducted by segmentation into many portions or by sampling [12]. In this paper, we chose several parsers that are currently popular and commonly used. They are SLCT [13], AEL [14], IPLoM [15], LKE [16], LFA [17], LogSig [18], SHISO [19], LogCluster [20], LenMa [21], LogMine [22], Spell [23], Drain [2], MoLFI [3].

The parsers used in the study have different algorithms and methods for log parsing. LogMine is a parser that extracts structured information from logs by using regular expressions and heuristics. It is capable of handling a wide range of log formats and can automatically learn new patterns from unlabeled data. Drain is a parser that groups similar log messages together using a log clustering algorithm and then applies a set of predefined templates to extract structured information from each cluster.

Spell is a parser that learns the structure of log messages using a probabilistic model and applies this model to parse new logs. It is designed to handle noisy and heterogeneous logs with varying levels of structure. AEL is a parser that combines multiple parsing techniques, including regular expressions, decision trees, and support vector machines, using an ensemble learning approach. It is designed to handle complex logs with multiple levels of structure. SLCT uses a frequency-based approach to identify constant and variable parts of log messages. It has low computational complexity but may generate inaccurate templates for logs with low frequency or variable length. IPLoM uses a partitioning-based approach to split log messages into groups based on length, token position, and token frequency. It can handle high-dimensional logs but may produce redundant or incomplete templates for complex logs.

MoLFI uses an evolutionary-based approach to finding the optimal set of templates for log messages based on multiple objectives. It can handle complex and diverse logs but may require high computational complexity and parameter tuning. LKE uses a clustering-based approach to group log messages based on string similarity and extract templates using longest common substrings. It can handle noisy logs but may require manual intervention to merge clusters or refine templates. LFA uses a factorization-based approach to decompose log messages into a dictionary of words and a matrix of coefficients. It can handle large-scale logs but may be affected by noise or outliers in the log data.

LogSig uses a signature-based approach to generate unique identifiers for log messages based on their punctuation and length features. It can handle online log parsing but may fail to capture semantic similarities or differences among log messages. SHISO uses a self-supervised learning approach to train a neural network to generate templates for log messages. It can handle complex and diverse logs but may require a large amount of training data and computational resources. LogCluster uses a density-based approach to cluster log messages based on word frequency and extract templates using the longest common substrings. It can handle variable-length and high-dimensional logs but may be sensitive to parameter settings and noise in the log data. LenMa uses a length-based approach to split log messages into groups based on length and extract templates using the longest common substrings. It can handle variable-length logs but may produce inaccurate templates for logs with similar lengths but different structures.

Our proposed parser uses a combination of data-based features and rules to extract structured information from logs. It is flexible and adaptable to different log formats and can be trained on labeled data. The study evaluated these parsers on multiple datasets using various metrics to identify their strengths and weaknesses and to gain insights into their relative performance.

Furthermore, there have been other proposed log parsers. Yu et al. [24] propose an approach and system framework to use documentation knowledge for log parsing. It can improve the parsing accuracy and discover the linkages between event templates. Rand et al. [25] explore the possibility of automating the parsing task by employing machine translation. It creates a tool that generates synthetic Apache log records and trains recurrent neural network-based models. It shows that the models can learn Apache log format and parse individual log records. Fastlogsim [26] realizes fast log parsing by calculating text similarity and merging similar templates. The paper [27] is mainly about the execution log of two or more systems, outputting a model of their differences. LogHound [28] mainly uses a frequent itemset mining algorithm to discover frequent patterns from event logs. To summarize, current log parsing technology is relatively mature, primarily through the definition of rules, code analysis, and machine learning methods to achieve rapid parsing of various log structures, removing irrelevant content, extracting key fields of various events, and storing information for later analysis.

B. Log Analysis

Log analysis is also commonly used for anomaly identification. The use of artificial intelligence systems to automatically examine system logs to uncover and diagnose errors is at the heart of log anomaly detection. A lot of studies have been done on existing log-based anomaly detection algorithms [29], [30], [31]. Manually inspecting logs to discover abnormalities becomes infeasible as the volume and complexity of existing systems grow. As a result, the necessity for automatic anomaly detection for log data is becoming increasingly critical, and several solutions in related disciplines have been presented [29], [32], [33]. The use of artificial intelligence systems to automatically examine system logs to uncover and diagnose errors is at the heart of log anomaly detection. Machine learning-related technologies have exploded in popularity, resulting in widespread adoption. LogClass is a data-driven system for detecting and categorizing anomalies in device logs [4]. LogClass builds bag-of-words vectors, then trains anomaly detection and classification models using the PU learning approach and SVM, and lastly conducts online anomaly detection. Semi-supervised algorithms work well with labeled data that is entirely normal. Yang et al. researched this [34]. Anomaly detection techniques based on deep learning are becoming increasingly popular, and their use in a range of problem sets outperforms previous methods [35]. LADRA [36] is an abnormality detection tool based on GRNN neural network model.

III. The Present

In this section, we discuss the experimental data, how preprocessing affects log parsing, how we measure the results, and what we found about parsing errors.

A. Datasets

The data format is versatile and adaptable. For parser evaluation, selecting adequate datasets is critical. We use the datasets and benchmarks gathered and proposed by Zhu et al. [2] in this study. All the log datasets are presented in Table I. Time duration, data size, message count, and template count are all included in the content. The log datasets come from a variety of sources, including distributed systems, operating systems, and mobile devices. HDFS [37] is a distributed file system under Hadoop. It is the lowest-level distributed storage service and is one of Hadoop's essential components. It's a storage system that connects several computers. Hadoop [38] is an Apache open-source Java framework for distributed data processing across clusters of computers utilizing a simple programming model. Spark [39] is an open-source platform for interactive inquiry, machine learning, and real-time workloads. It does not have own storage system and will instead rely on HDFS and other storage systems. Spark processes data in RAM utilizing a concept known as a Resilient Distributed Dataset, while Hadoop reads and writes files to HDFS. BGL [40] is a powerful supercomputer led by IBM. MacOS [41] is a graphical user interface-based operating system launched by Apple and is the main operating system for Macintosh series computers. Microsoft Windows [42] is a family of proprietary commercial software operating systems with a graphical user interface developed by Microsoft. HealthApp [3] is a mobile application that collects and manages personal health information. Proxifier [11] is a sophisticated socks5

[3]https://github.com/logpai/loghub/tree/master/HealthApp

client that allows network programs that don't support proxy servers to pass HTTPS or SOCKS proxies or proxy chains. Data from different systems can better assess the generality of the parsers, and the outcomes of the experiments are also more persuasive.

B. *Preprocessing*

The following are the primary goals of log analysis data preprocessing: (1) Filter out the non-compliant data and remove the meaningless data. (2) Regularization and format conversion. (3) Filter and segregate the various data columns according to the later use requirements.

In this research, we use a log data preparation approach. To make the parsers' processing easier, we first filter the data, then create rules to replace parameters in log data with regular expressions. To further understand the impact of preparation on log parsing, we go through three steps.

- We evaluate the selected log parsers without using any preprocessing algorithms.
- We filter the data without using regular expression techniques.
- We use regular expressions after filtering all log data.

Data filtering. As shown in Fig. 1, there are generally multiple columns in log data, such as Date, Time, ID, and Component. Templates, on the other hand, are just concerned with the content. The content element is crucial for log parsers, while the other parts are less important. We only preserve the content column in this section and delete the other columns.

Regular expression. Regular expressions are string-matching patterns that can be used to check if a string contains a specific substring, replace a matching substring, extract a substring that meets a specific condition from a string, and so on. As a result, we use the symbol <*>to replace the specified strings. We developed the following broad string selection principles based on our observations of experimental data:

- All strings with numbers in them (date, IP, numbers, hexadecimal numbers etc.).
- URLs.
- Strings to the right of the equal sign.
- Paths.
- String after special nouns (e.g., user, admin)

The examples of rules can be found in Table II. The broad guidelines, however, cannot cover all data.

TABLE II
EXAMPLES OF RULES.

Log data	Regular expression
Register: 0x0002f900	Register: <*>
URL: https://tools.google.com/service/update2?	URL: <*>
productID= com.google.Keystone	productID= <*>
file /home/pakin1/sweep3d-2.2b/results/random1-8x32x32x2.map	file <*>
user admin	user <*>

In this paper, we choose 13 existing log parsers. Table III shows the effective log parsing tools. Some of these methods, such as Drain and Spell, use preprocessing, while others do not. If a log parsing technique includes a pretreatment phase, we mark it with a ✓, and if it does not, we mark it with an ✗. These parsers cover a wide range of methodologies and can represent the current industry standard, making them useful research tools. Other research work [2] [5] have also utilised these log parsers. A log keeps track of a series of events in a system or application. A record usually represents the content of an event. The time, source, tags, and information are often contained in log records, but they are not the only ones. In most cases, log parsing entails evaluating log records, deleting unnecessary information, and then understanding the log content using related methods.

TABLE III
SUMMARY OF AUTOMATED LOG PARSERS.

Parser	Year	Preprocessing
SLCT	2003	✗
AEL	2008	✓
IPLoM	2012	✗
LKE	2012	✓
LFA	2010	✗
LogSig	2011	✗
SHISO	2013	✗
LogCluster	2015	✗
LenMa	2016	✗
LogMine	2016	✓
Spell	2016	✓
Drain	2017	✓
MoLFI	2018	✓

C. *Evaluation Metrics*

Different definitions for correctly processed log messages and templates are used in the existing accuracy measures. The following are the metrics that were picked to address the issue in the section.

Grouping Accuracy (GA). The GA metric [5] is used to assess the accuracy of log template identification, we follow the name proposed by khan et al. It's calculated as the proportion of correctly parsed log messages to total log messages. Each log message has an event template after it has been parsed, which corresponds to a set of messages with the same template [2]. The GA metric does not consider whether the identified templates are the same as the ground trues but only considers if they are grouped correctly.

Parsing accuracy (PA). The PA metric [8] is similar to GA in that it measures the proportion of correctly processed log messages to total log messages. It is more stringent than GA because it takes into account static text and variables. In Table IV, we offer an example of a comparison of different metrics. The sample set of logs messages are M= m_1,m_2, m_3,...m_n. The obtained templates after parsing are represented as T= t_1,t_2, t_3,...t_n, and the corresponding templates of the ground truth are G= g_1,g_2, g_3,...g_n. We use ✓if the parsing result is determined to be correct; otherwise, we use ✗. The metric GA of m_4 and m_8 are correct, while PA are wrong. We can deduce that if PA=1, then GA=1, but not the other way around. GA gets a 4/7 because five out of seven messages (m_1,m_2, m_3, m_7) can be correctly parsed under the GA criterion; PA gets a 3/7 because four out of seven messages (m_1,m_2, m_3) can be correctly parsed under the PA criterion.

Template Accuracy (TA) Metrics. To test the successfully parsed templates considerably more carefully, the TA metrics [5] are proposed. These are the most stringent measures in comparison to the others. There are more variables to consider. Khan et al. introduce two TA metrics: **Precision-TA (PTA)** and **Recall-TA (RTA)**. On the basis of the metrics, we add one more metric **F1 score-TA (F1TA)** which is dependent on the metrics PTA and RTA and can be calculated by them. The TA metrics not only consider the correctness of template classification but also the correctness of template content. In Table IV, the correctly identified templates are t_1 and t_2, then the PTA can be calculated as 2/5. The RTA also can be calculated as 2/5 because two templates (g_1, g_2) out of five are correct. There are variances between the metrics, as can be shown.

$$F_1TA = \frac{2PTA * RTA}{PTA + RTA} = \frac{2TP}{2TP + FP + FN} \qquad (1)$$

In this paper, we selected five evaluation metrics, namely PA, GA, PTA, RTA, and FTA, which are commonly used in previous log parsing research. Each metric has unique characteristics, as well as its own strengths and weaknesses. GA is a straightforward metric that measures how well a log parser groups or clusters log messages. It is useful for understanding log structures and detecting potential issues or anomalies. It is particularly helpful for evaluating parsers that cluster or group log messages. However, it only considers the grouping of log messages and not their content, which may result in inaccurate results when identifying specific log events or issues. It also assumes that the ground truth labeling or clustering is correct and may not be sensitive to differences in the distribution or prevalence of different log categories or clusters, which can impact its performance in real-world situations.

PA is a commonly used metric in log parsing that measures the proportion of accurately parsed messages in a log file. Its strengths include being simple to understand and widely used in research, making it useful for comparing parser performance. However, it has limitations, such as not considering the extracted message's content, being sensitive to the quality of the ground truth labels, and not accounting for parsing performance on a per-line basis. As such, it should be used alongside other metrics and qualitative analysis to fully assess a parser's performance.

TA is a metric that evaluates a log parser's ability to identify and extract log message templates based on their similarity to the ground truth templates. It is particularly useful for parsers using template-based approaches and provides insights into the parser's ability to generalize across different log files. However, it requires a ground truth template set, which may not always be available, and does not evaluate the parsing accuracy of log messages that do not match any ground truth template. Additionally, it may not accurately reflect the quality of the log parser's output in terms of identifying specific log events or issues, so it should be used in conjunction with other metrics and qualitative analysis for a complete evaluation of the parser's performance.

Overall, GA, PA, and TA are commonly used metrics for evaluating the performance of log parsers because they provide different perspectives on the performance of a parser in different aspects of log parsing. By evaluating parsers using these three metrics, we can gain a comprehensive understanding of the performance of the parser in different aspects of log parsing, including message template extraction, parsing accuracy, and grouping accuracy.

D. *Experiments*

We use 13 existing log parsing approaches on 16 benchmark datasets in this part to conduct various experiments on the difficulties outlined above. The benchmarking results are presented in terms of three research questions.

- **RQ1**: How do the existing approaches perform on different metrics?
- **RQ2**: How likely would preprocessing improve the performance?
- **RQ3**: How do the existing approaches perform under different preprocessing conditions?

To answer RQ1, we compare and contrast the 13 log parsers on 16 datasets using various metrics (GA, PA, PTA, RTA, and F1TA). The test environment and equipment types employed in each parser's evaluation process are the same. The findings of each parser were averaged across all datasets and are displayed in the first column of Table V. Since different assessment measures have distinct criteria, the outcomes of the same parser under various metrics are quite different. Table V shows experimental findings ranging from 0.07 to 0.87. The Drain parser receives the highest score and the LogSig parser receives the lowest score in the GA metric; however, this does not imply that the same is true for other metrics. The SLCT and Drain parsers receive the greatest PA scores, while the MoLFI parser receives the lowest. Actually, the experimental findings of the same parser on the same dataset vary dramatically. We used Drain as an example and checked each measure on the HDFS dataset; the result of GA is 1, but all other metrics are 0. Furthermore, the results of multiple datasets for the same metric indicate significant differences. Drain, for example, is used to analyze all datasets using the GA measure. Although Apache's result is 1, Proxifier's GA score is only 0.51. Despite the fact that the ratings vary greatly, the Drain parser consistently ranks first in each category.

Since different assessment measures have distinct criteria, the outcomes of the same parser under various metrics are quite different. This means that some parsers may perform well on one metric but poorly on another metric. As a result,

TABLE IV
AN EXAMPLE OF COMPARISON OF DIFFERENT METRICS.

Message	Template	Ground truth	GA	PA
(m_1) PacketResponder 1	(t_1) PacketResponder <*>	(g_1) PacketResponder <*>	✓	✓
(m_2) PacketResponder 0	(t_1) PacketResponder <*>	(g_1) PacketResponder <*>	✓	✓
(m_3) blk-12330058	(t_2) blk- <*>	(g_2) blk- <*>	✓	✓
(m_4) blk-8736461	(t_3) <*>	(g_2) blk- <*>	✗	✗
(m_5) updated: 10:50010	(t_4) updated: <*>	(g_3) updated: <*>: <*>	✗	✗
(m_6) updated: 10:50010:46509	(t_4) updated: <*>	(g_4) updated: <*>: <*>: <*>	✗	✗
(m_7) route = 0x0	(t_5) <*>= 0x0	(g_5) route = <*>	✓	✗

TABLE V
RESULTS OF LOG PARSERS ON DIFFERENT DATASETS.

	Original parsing results					Parsing without preprocessing					Parsing with data filtering					Parsing with regular expressions				
Parser	GA	PA	PTA	RTA	F1TA	GA	PA	PTA	RTA	F1TA	GA	PA	PTA	RTA	F1TA	GA	PA	PTA	RTA	F1TA
SLCT	0.64	0.29	0.22	0.16	0.19	0.64	0.29	0.22	0.16	0.19	0.64	0.30	0.22	0.16	0.19	0.64	0.31	0.23	0.18	0.20
AEL	0.75	0.25	0.25	0.27	0.26	0.72	0.21	0.21	0.23	0.22	0.75	0.26	0.25	0.27	0.26	0.75	0.25	0.25	0.27	0.26
IPLoM	0.77	0.19	0.16	0.15	0.15	0.77	0.19	0.16	0.15	0.15	0.75	0.19	0.16	0.16	0.16	**0.73**	**0.10**	**0.11**	**0.10**	**0.11**
LKE	0.56	0.12	0.12	0.14	0.13	**0.34**	**0.06**	**0.09**	**0.07**	**0.07**	**0.34**	**0.06**	**0.09**	**0.06**	**0.06**	0.53	0.08	0.16	0.11	0.13
LFA	0.65	0.20	0.14	0.14	0.14	0.65	0.20	0.14	0.14	0.14	0.65	0.21	0.14	0.15	0.14	0.65	0.16	0.16	0.17	0.16
SHISO	0.68	0.13	0.16	0.17	0.16	0.68	0.13	0.16	0.17	0.16	0.68	0.13	0.16	0.17	0.16	0.68	0.13	0.16	0.17	0.16
LogSig	0.48	0.12	0.08	0.07	0.07	0.48	0.12	0.08	0.07	0.07	0.48	0.12	0.08	0.07	0.07	0.53	0.15	0.10	0.08	0.09
LogCluster	0.67	0.14	0.11	0.22	0.15	0.67	0.14	0.11	0.22	0.15	0.67	0.15	0.11	0.22	0.15	0.67	0.18	0.15	0.22	0.18
LenMa	0.72	0.19	0.17	0.23	0.20	0.72	0.19	0.17	0.23	0.20	0.72	0.19	0.17	0.23	0.20	**0.83**	**0.24**	**0.30**	**0.30**	**0.29**
LogMine	0.69	0.22	0.16	0.20	0.18	**0.31**	**0.15**	**0.05**	**0.23**	0.07	0.31	0.15	0.05	0.23	0.07	**0.69**	**0.22**	**0.16**	**0.20**	**0.18**
Spell	0.75	0.19	0.15	0.17	0.16	0.70	0.14	0.14	0.14	0.13	0.75	0.19	0.15	0.17	0.16	0.75	0.19	0.15	0.17	0.16
Drain	0.87	0.29	0.27	0.29	0.28	0.87	0.25	0.25	0.26	0.25	0.87	0.25	0.25	0.26	0.25	0.87	0.29	0.27	0.29	0.28
MoLFI	0.61	0.11	0.09	0.09	0.09	0.63	0.08	0.09	0.11	0.09	0.60	0.09	0.08	0.10	0.09	0.59	0.11	0.09	0.09	0.09

we observed that there is significant variation in parser performance under different evaluation metrics. Therefore, while some parsers may perform well overall, their performance may be inconsistent across different metrics. This highlights the importance of carefully selecting appropriate evaluation metrics when comparing log parsers and interpreting their results.

Summary The results of some metrics are very inconsistent. They give different results when used on different types of data.

To answer RQ2 and RQ3, in this section, we look at how preprocessing affects parsers in this part. The tests are separated into four parts, as previously stated: (1) All parsers with no preprocessing, (2) All parsers with only data filter but no regular expressions, and (3) All parsers with both data filter and regular expressions. The experimental results are displayed in the second, third, and fourth columns of Table V respectively. We notice that there is a very small gap between the results in the second and third columns. Many of the outcomes are nearly identical. It can therefore be deduced that data filtering has little impact on parsers. On the other hand, the obvious difference reflected in the third and fourth columns indicates that regular expressions have a certain impact on the parsers. Preprocessing has varying effects on different parsers. We highlight some obvious examples in Table V. Details of them are described in the following paragraph.

The LogMine parser itself comes with preprocessing. When the preprocessing part is deleted, the results drop significantly. The GA score dropped from 0.69 to 0.31; the PA score changed from 0.22 to 0.15; the F1TA score dropped from 0.18 to 0.07; the scores of other metrics fall to some extent. The LKE parser behaves similarly. It is assumed that these two parsers both rely on regular expressions to some extent. Preprocessing can also greatly enhance the outcomes of some other parsers, such as the LenMa parser. The GA score went up from 0.72 to 0.83; the PA score went up from 0.19 to 0.24; the F1TA score went up from 0.20 to 0.29. However, not all parsers benefit from preprocessing. The impact is negative for the IPLoM parser. When preprocessing is used, the GA score drops from 0.77 to 0.75. The construction of regular expressions does not encompass all data; the exceptions are just disregarded. If the data formats and contents are consistent, preprocessing may have good consequences, and general regular expressions can cover all or most of the data; on the other hand, if the data formats and contents differ, and numerous special circumstances are present, preprocessing

TABLE VI
RESULTS OF PARSERS ON DIFFERENT TEMPLATE FORMS. "P" REFERS TO "PARAMETER".

Parser	Start with P	End with P	continuous P	Overall accuracy
IPLoM	5.99% (230/3837)	14.18% (2345/16,543	1.71% (11/643)	18.58% (5944/32,000)
AEL	6.10% (234/3837)	18.27% (3022/16,543)	3.11% (20/643)	25.72% (8229/32,000)
Spell	5.99% (230/3837)	14.18% (2158/16,543)	0% (0/643)	18.30% (5855/32,000)
Drain	6.18% (237/3837)	20.81% (3443/16,543)	24.41% (157/643)	29.18% (9388/32,000)
LenMa	0.10% (4/3837)	3.27% (541/16,543)	3.27% (21/643)	17.68% (5658/32,000)
LFA	5.99% (230/3837)	16.33% (2701/16,543)	0.47% (3/643)	19.51% (6243/32,000)
LKE	0% (0/3837)	0% (0/16,543)	0% (0/643)	8.63% (2762/32,000)
LogMine	6.07% (233/3837)	13.37% (2211/16,543)	20.99% (135/643)	20.34% (6509/32,000)
LogSig	0% (0/3837)	0% (0/16,543)	0% (0/643)	12.67% (4054/32,000)
MoLFI	0% (0/3837)	0% (0/16,543)	0% (0/643)	9.04% (2892/32,000)
SHISO	0.10% (4/3837)	2.13% (353/16,543)	1.71% (11/643)	12.72% (4069/32,000)
SLCT	4.80% (184/3837)	18.43%(3049/16,543)	3.58% (23/643)	30.31% (9700/32,000)
LenMA	0.10% (4/3837)	3.27% (541/16,543)	3.27% (21/643)	17.68% (5658/32,000)

may have negative impacts.

The reason for this could be that some log messages contain complex or irregular patterns that are difficult to capture using regular expressions or other preprocessing techniques. In such cases, the parser may not be able to accurately extract all the relevant information from the log message, even with preprocessing. Even if a preprocessing technique is designed for a specific log format, errors can occur during the preprocessing stage. For example, if the preprocessing technique relies on a regular expression to extract relevant information, the expression may not match all instances of the pattern in the log data, resulting in incomplete or inaccurate preprocessing. Another possible reason is that some parsers may already have built-in preprocessing techniques that are optimized for certain types of log messages, such as Drain. In such cases, additional preprocessing may not provide any significant improvement in performance. Apart from the dimension of accuracy, preprocessing can initially filter the data, decreasing the load on the later parser, reducing runtime, and therefore improving the parser's efficiency.

Summary (1) Preprocessing usually helps most parsers, it can boost the GA value by up to 0.38 and the PA value by up to 0.07. But there are some exceptions where it does not work. (2) Regular expressions are mainly used in preprocessing, but data filtering is not.

E. Incorrect Parser Results

The trials mentioned above show that the outcomes for several metrics are not optimal. The outcomes of the experiment have been examined. We evaluate the outcomes of the experiment in an effort to identify the problem and its cause. The current parsers do a good job of classifying log templates, but it needs to parse log templates better. We choose several parsers for analysis. In parallel, a number of template forms are chosen to gauge the parser's effectiveness. The results under examination are (1) The proper parsing rate (percentage of correctly parsed templates) of templates that start with a parameter, (2) The proper parsing rate of templates that end with a parameter, (3) The proper parsing rate of templates that contain continuous parameters, (4) The size of the templates' parameters list, and (5) The average distance between parameters in templates. We examine the parsers using 16 datasets and show the summarized results in Table VI and Fig 2.

When parsing templates with successive parameters, most parsers are ineffective, except Drain and LogMine. Most parsing templates with successive parameters have a fixed order of parameters, which makes them ineffective in handling the high variability and complexity of log messages. In other words, the templates require a certain order of parameters to extract information, but log messages often have inconsistent or varied orders of parameters. This can result in parsing errors and make it difficult to extract structured information from logs. However, Drain and LogMine are effective in handling this issue because they use log clustering and regular expressions, respectively, to extract structured information from logs. They do not rely on a fixed order of parameters and can handle logs with varying parameter orders. Additionally, the Proxifier dataset basically starts with "c.baidu.com:80" and "proxy.cse.cuhk.edu.hk:5070", making it very difficult for IPLoM to recognize them, which results in very low performance.

Diverse datasets produce wildly different parsing results. AEL, Drain and LenMa score 100% on the Android dataset using the "Start with parameter" metric, while other datasets do not perform as well. This is most likely caused by the quality and format of the data. Fig 2 shows the analytical outcomes for different log message categories: for the number of tokens, we define six ranges, which are 1-2, 3-4, 5-6, 7-8, 9-10, and more than 11; for the number of parameters, the ranges are 0, 1, 2, 3, 4, and over 5; for the average

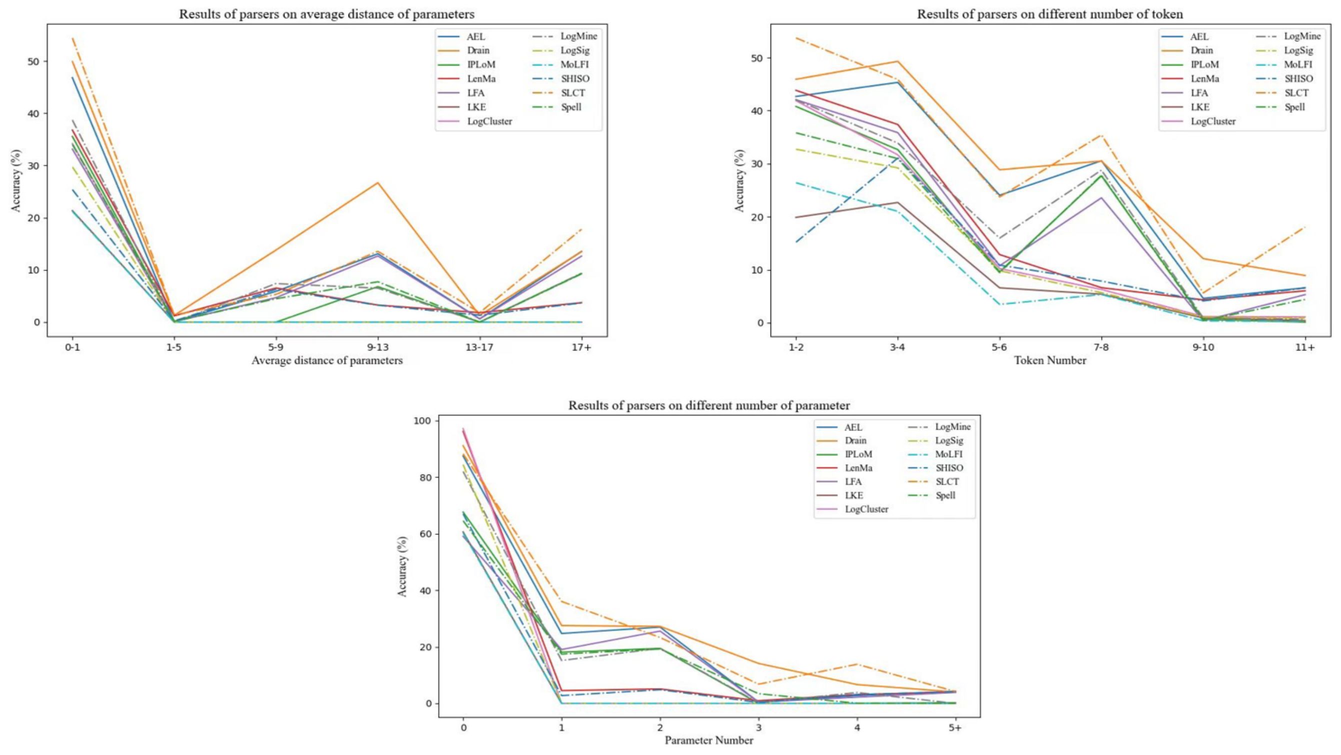


Fig. 2. Results of parsers on different template forms.

distance of parameters, the ranges are 0-1, 1-5, 5-9, 9-13, 13-14, and over 17. The fewer the number of tokens or parameters a parser deals with, the better accuracy it can achieve. When the parameters in the log message have an average distance of zero (which is actually the case where the number of parameters is 0 or 1), the parser performs better. When there are parameters in the log template, LKE and LogSig cannot recognize them. The paper only summarizes the results, while more thorough and in-depth findings are put in GitHub.[4] Here, we list a few instances of parsing mistakes from Drain in Table VII. Our finding is that Drain has poor parsing of numeric classes, symbols, etc. Besides, it occasionally produces different parsing results for the same log due to the fact that the context content differs. However, we found that several bugs were brought on by wrongly labeled Ground Truths. As a result, we decided to entirely and thoroughly modify the Ground Truth files.

Summary (1) Most parsers do not work well with log messages that have consecutive variables. (2) The parsers perform better when there are fewer tokens or parameters.

IV. The Future

In this section, we talk about how we changed the ground truth and made a regex-based parser. We also show the results of our tests to check how well they work.

[4]https://github.com/logpai/logparser/tree/master/logsrevised

A. Datasets

Multiple errors in the current Ground Truth: (1) There are some strings that are misrepresented as arguments. (2) Poor consistency. Symbols are handled poorly; some are assigned as parameters while others are not. (3) Poor interpretability. Certain parameter settings are absurd. According to a set of rules, we modify the Ground Truth files. For the logs that can be confirmed by the source code, we alter the files in accordance with the source code. For other logs, we aim for consistency and common sense. If the guidelines are not applicable, we use our best judgment. To keep the consistency, the following regulations have been created. (1) Combining letters, integers, and symbols into a single parameter, if there is no space in between, symbols are not preserved. However, the symbol must be kept if there is a space between it and the character. Whatever the case, parentheses, square brackets, curly brackets, etc. need to be preserved. (2) The following are regarded as parameters: pure numbers, numbers only mixed with decimal points, and numbers only mixed with letters. (3) Strings with numerous decimal points between them are regarded as parameters (the number of decimal points is greater than 2). (4) It is treated as a parameter if it comes after the equal sign. (5) Under some circumstances, capital letters are parameters (e.g., from INITED to SETUP). (6) Paths and URLs are regarded as parameters. Whenever a string has more than two slashes, it is treated as a URL or path. (7) The use of certain special terms as parameters, such as null, true, and false. (8) Strings are treated as parameters if they follow certain specific terms, such as user and admin. In Table VIII, we provide various samples. The majority of the log content

TABLE VII
INSTANCES OF PARSING MISTAKES FROM DRAIN.

Content	Result from Drain	Original Ground Truth	What we deem to be the proper Ground Truth
Starting Socket Reader #1 for port 62260	Starting Socket Reader #1 for port <*>	Starting Socket Reader #<*>for port <*>	Starting Socket Reader <*>for port <*>
Scheduled snapshot period at 10 second(s).	Scheduled snapshot period at 10 second(s).	Scheduled snapshot period at <*>second(s).	Scheduled snapshot period at <*>second(s).
adding path spec: /mapreduce/*	adding path spec: <*>	adding path spec: /<*>/	adding path spec: <*>

TABLE VIII
EXAMPLE OF MODIFYING GROUND TRUTH TEMPLATES.

Content	Original template	Modified template
appattempt_1445144423722_0020_000001	appattempt_<*>	<*>
org.apache.hadoop.mapreduce.lib.output.File	org.apache.hadoop.mapreduce.lib.output.File	<*>
OutputCommitter set in config null	OutputCommitter set in config null	OutputCommitter set in config <*>
Transitioned from NEW to INITED	Transitioned from NEW to INITED	Transitioned from <*>to <*>
http://wiki.apache.org/hadoop/NoRouteTo	http://wiki.apache.org/hadoop/NoRouteTo	<*>
admin [preauth]	admin [preauth]	user [<*>]
Function=int	Function=int	Function=<*>
fpr29=0xffffffff ffffffff ffffffff ffffffff	fpr29=<*>	fpr29=<*><*><*><*>

is processed using these rules. We have not made any changes for some particular circumstances or issues with the original data. After adjustment, the type and quantity of templates will change, and the specifics will be shown in Table IX. We provide some details, such as how many template types have been edited, the number of original template types, and how many template types existed.

TABLE IX
MODIFICATION OF THE GROUND TRUTH TEMPLATE TYPES.

Dataset	Modified	Original	Now
HDFS	14	14	14
Hadoop	77	114	103
Spark	14	36	32
ZooKeeper	17	50	49
OpenStack	14	43	43
BGL	29	120	120
HPC	11	46	44
Thunderbird	58	149	149
Windows	12	50	49
Linux	48	118	115
Mac	161	341	324
Android	76	166	155
HealthApp	21	75	73
Apache	1	6	6
OpenSSH	19	27	27
Proxifier	15	8	15

B. *Regular Expressions*

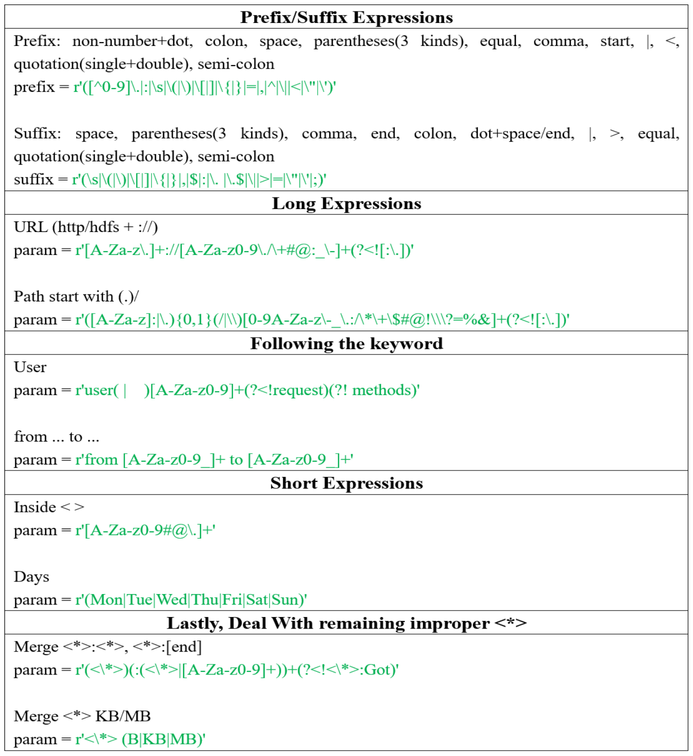
Since the existing one performs poorly when parsing some elements of the log content, as khan et al. [5] also point out that some metrics do not favor existing approaches, in this section, we intend to present a straightforward and efficient parser. The regex-based parsers mentioned by Zhu et al.[2] rely on the manual construction of regexes for matching all possible logs, which could be hundreds of complex regexes (e.g., 120 regexes for the 2k BGL dataset). Differently, the regexes in this paper are designed to match tokens (instead of logs). Thus, we only need to manually construct several simple regexes for matching tokens in all datasets. We use CSV files for both input and output. We created a regular expression-based parser according to the modified Ground Truth rules and the log content. Here, we provide some pseudocode.

It has 23 regexes that match common log patterns, such as timestamps, process IDs, and error codes, and they fall approximately into five groups. We chose these groups based on our data and they can help us start developing regexes for log parsing. The first group has strings with numbers, like dates, IP addresses, and hex numbers. These strings can tell us when and where events happened. The second group has URLs, which are common in web server logs and can tell us what pages users visited on a website. The third group has strings after an equal sign in a log message. These strings often assign values to variables or parameters and can help us identify specific events or actions. The fourth group has paths, which often show file locations or directories in a log message. These strings can help us identify specific files or directories that were used or changed. The last group has strings after special nouns like "user" or "admin". These strings can help us identify specific users or administrators who did certain actions in a log message. These groups are not complete and may not cover all data in log parsing. But they can help us start making regexes and grouping information for log analysis. Some regular expressions are displayed in Fig. 3. The attachment contains all more information. Using both the original Ground Truth files and the modified Ground Truth files, we test the existing parsers and our parser. The benchmarking results are presented in terms of three research questions.

- **RQ4**: How do the existing approaches perform on the new Ground Truth? How is this performance different from that on the original Ground Truth?
- **RQ5**: How does our regex-based parser perform under different Ground Truths and metrics?

```
Input :
  $C = \{c_1, c_2, ..., c_x\}$: log contents
  $I = \{i_1, i_2, ..., i_y\}$: regular expressions for identifying a target string in
    the log content
  $S = \{s_1, s_2, ..., s_y\}$: regular expressions to extract a substring from the
    identified string
  $R = \{r_1, r_2, ..., r_y\}$: strings used to replace the substrings
  where $x = |C|$, $y = |I| = |S| = |R|$.
Function fitRE($C$, $I$, $S$, $R$):
    $C' = list()$
    // For each log
    for $m$ from 1 to $x$ do
        // For each RE rule
        for $n$ from 1 to $y$ do
            // Perform replacement until the search
                fails
            while re.search($i_n$, $c_m$) do
                // Search for $i_n$ in $c_m$
                $idfString$ ← re.search($i_n$, $c_m$)
                // Search for $s_n$ in $idfString$
                $subString$ ← re.search($s_n$, $idfString$)
                // Replace $subString$ by $r_n$ in
                    $idfString$
                $repString$ ← idfString.replace($subString$, $r_n$)
                // Replace $idfString$ by $repString$ in
                    $c_m$
                $c_m$ ← $c_m$.replace($idfString$, $repString$)
            end
        end
        $C'$.append($c_m$)
    end
    return $C'$
```

Algorithm 1: Pseudocode of conducting the regular expressions



Prefix/Suffix Expressions

Prefix: non-number+dot, colon, space, parentheses(3 kinds), equal, comma, start, |, <, quotation(single+double), semi-colon

```
prefix = r'([^0-9]\.|:|\s|\(|\)|\[|]|\{|}|=|,|^|\||<|\"|\')'
```

Suffix: space, parentheses(3 kinds), comma, end, colon, dot+space/end, |, >, equal, quotation(single+double), semi-colon

```
suffix = r'(\s|\(|\)|\[|]|\{|}|,|$|:|\. |\.$|\||>|=|\"|\'|;)'
```

Long Expressions

URL (http/hdfs + ://)

```
param = r'[A-Za-z\.]+://[A-Za-z0-9\./\+#@:_\-]+(?<![:\.])'
```

Path start with (.)/

```
param = r'([A-Za-z]:|\.){0,1}(/|\\)[0-9A-Za-z\-_\.:/\*\+\$#@!\\\?=%&]+(?<![:\.])'
```

Following the keyword

User

```
param = r'user( |    )[A-Za-z0-9]+(?<!request)(?! methods)'
```

from ... to ...

```
param = r'from [A-Za-z0-9_]+ to [A-Za-z0-9_]+'
```

Short Expressions

Inside < >

```
param = r'[A-Za-z0-9#@\.]+'
```

Days

```
param = r'(Mon|Tue|Wed|Thu|Fri|Sat|Sun)'
```

Lastly, Deal With remaining improper <*>

Merge <*>:<*>, <*>:[end]

```
param = r'(<\*>)(:(<\*>|[A-Za-z0-9]+))+(?<!<\*>:Got)'
```

Merge <*> KB/MB

```
param = r'<\*> (B|KB|MB)'
```

Fig. 3. Formulation of regular expressions.

Under the original Ground Truth and the new Ground Truth, we list the ranking of parsers for each metric in Table X. The ranking of parsing results under each metric has changed using different Ground Truths. It is clear that the change to Ground Truth has an influence on the current parsers, whether it has a positive or negative impact. The RTA metric has resulted in a major change in the LogCluster rankings; RTA is ranked 4 with the original Ground Truth but dips to 11 with the new Ground Truth. Similar things arise when using different parsers and metrics. Table XI displays all of the outcomes. In Table XI, we present the evaluation outcomes of the regex-based parser for the initial Ground Truth and the current Ground Truth, respectively. We list the top outcomes from other parsers in the final row of Table XI. Except for the value of GA, our regex-based parser outperforms all other parsers in terms of other metrics based on the original Ground Truth. Khan et al. [5] point out the ineffectiveness of current parsers and how poorly they perform on their proposed metrics (PTA and RTA). However, even with the original Ground Truth, our regex-based parser can still produce good results across the metrics. The regex-based parser's performance has significantly improved since utilizing the new Ground Truth. The results of all metrics outperform those of all other parsers and are near to being full marks. Despite not utilizing machine learning, our parser can nevertheless produce excellent results.

To confirm the versatility of our parser, we gathered log data from the CodeArts PerfTest system of Huawei company. CodeArts PerfTest is a cloud-based service that enables performance testing for application interfaces and links. It supports protocols such as HTTP/HTTPS/TCP/UDP and offers a wide range of test model definition capabilities to simulate real-world business access scenarios. By identifying application performance problems beforehand, users can make necessary improvements and avoid potential issues. We utilized it in our experiments to investigate the generalizability of our approach. Without writing any additional regular expressions, we directly applied the regex-based parser to industrial data and achieved impressive results, outperforming other parsers. Our parser performed exceptionally well, with a PA value of 0.98, surpassing other parsers by 0.88. Additionally, the RTA reached 0.9778, outperforming other parsers by 0.8667.

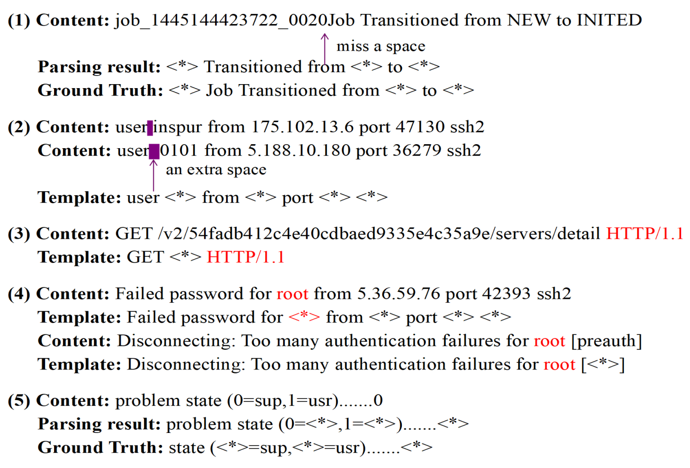
TABLE X
THE RANKING RESULTS OF LOG PARSERS ON ORIGINAL AND NEW GROUND TRUTH.

Parser	Rank	GA	PA	PTA	RTA	F1TA
SLCT	Original	10	1	3	8	4
	New	11	3	3	8	5
AEL	Original	3	3	2	2	2
	New	2	2	2	2	2
IPLoM	Original	2	6	6	9	9
	New	3	7	6	9	8
LKE	Original	12	11	10	11	11
	New	12	13	12	12	12
LFA	Original	9	5	**9**	10	10
	New	8	4	**5**	7	7
SHISO	Original	7	10	9	6	6
	New	9	9	5	7	7
LogSig	Original	13	12	13	13	13
	New	13	11	13	13	13
LogCluster	Original	8	9	11	**4**	8
	New	10	10	10	**11**	11
LenMa	Original	5	**4**	**4**	5	3
	New	4	**8**	**8**	3	4
LogMine	Original	6	4	**5**	5	5
	New	6	5	**9**	4	9
Spell	Original	4	7	**8**	7	7
	New	5	6	**4**	6	3
Drain	Original	1	1	1	1	1
	New	1	1	1	1	1
MoLFI	Original	**11**	13	12	12	12
	New	**8**	12	11	10	10

Summary (1) Changing the Ground Truth files affects the current parsers, both good and bad. The rankings of the results can change by up to 7. (2) Parser technology is not bad. Our regex-based parser can still produce good results using the original Ground Truth across different metrics. (3) Our regex-based parser has improved a lot with the modified Ground Truth, and now all the metric evaluation values are almost perfect.

V. THE UNCERTAINTIES

In our work, there are various uncertainties that mostly stem from two aspects. (1) Some log templates are too complex for the current parsers to handle correctly, or too difficult to verify their correctness. The dataset itself has a number of issues that the parser finds challenging to identify and address. Multiple spaces or missing spaces frequently occur in logs, which has significant negative influence on log parsing. We give some examples in Fig. 4. Due to the absence of space, the parser will interpret ”job_1445144423722_0020Job” as a whole parameter in the first example. The recognized template



(1) Content: job_1445144423722_0020Job Transitioned from NEW to INITED
miss a space
Parsing result: <*> Transitioned from <*> to <*>
Ground Truth: <*> Job Transitioned from <*> to <*>

(2) Content: user inspur from 175.102.13.6 port 47130 ssh2
Content: user 0101 from 5.188.10.180 port 36279 ssh2
an extra space
Template: user <*> from <*> port <*> <*>

(3) Content: GET /v2/54fadb412c4e40cdbaed9335e4c35a9e/servers/detail HTTP/1.1
Template: GET <*> HTTP/1.1

(4) Content: Failed password for root from 5.36.59.76 port 42393 ssh2
Template: Failed password for <*> from <*> port <*> <*>
Content: Disconnecting: Too many authentication failures for root [preauth]
Template: Disconnecting: Too many authentication failures for root [<*>]

(5) Content: problem state (0=sup,1=usr).......0
Parsing result: problem state (0=<*>,1=<*>).......<*>
Ground Truth: state (<*>=sup,<*>=usr).......<*>

Fig. 4. Examples of uncertainties.

of the second example is different from the correct template because the content has more space. Also, the regex-based parser does not work well with dates, strings before and after equal signs (the fifth example). Additionally, there are particular regex rules that clash.

(2) It can be challenging to determine whether certain log templates are accurate while updating Ground Truth files. ”HTTP/1.1” should be a parameter in the third example if our rules are followed, however, we believe that it should not be in this instance, since a great number of log messages hold this term unchanged while varying in other parts. It is quite difficult to determine logs like this. In addition, various logs may interpret the same string differently; in the fourth illustration, the root may or may not be the parameter. This is quite difficult for the parser to handle; consequently, more rules are required to construct a more comprehensive and accurate regex parser.

VI. THREATS TO VALIDITY

The validity of our study may be subject to some threats. The selection of the log datasets poses the biggest risk of external validity. We reviewed the literature on log analysis and identified 16 datasets that are widely used in the relevant papers. On 16 general datasets, all of the existing parsers and our proposed parser are compared. However, the data has not been updated in a while, and both the Ground Truth files and the datasets contain numerous inaccuracies. We completely altered the Ground Truth files to make this better. The analysis in RQ4 and RQ5 demonstrates how the accuracy of the Ground Truth affects the evaluation outcomes. Threats to internal validity are factored into our methodology and may affect the parsing results. Our parser is generated according to data-based features and rules, the regex-based parser may not be suitable for other datasets. Expanding the scope of the study to include more evaluation metrics could result in a more comprehensive understanding of log parsing performance. To mitigate this threat, we try to make general rules to cover more types of log content. Besides, we gathered 2000 log data from the system of Huawei company and utilized it in our experiments to investigate the generalizability of our

TABLE XI
RESULTS OF REGEX-BASED PARSING.

	Parsing with Original Ground Truth					Parsing with new Ground Truth				
Dataset	GA	PA	PTA	RTA	FTA	GA	PA	PTA	RTA	FTA
HDFS	0.9975	0	0	0	0	0.9975	0.9985	0.88	1.0	0.93
Hadoop	0.88	0.158	0.2636	0.2543	0.2589	0.92	0.9195	0.8545	0.9038	0.8785
Spark	0.9965	0.7765	0.4643	0.52	0.4906	0.999	0.9995	0.9677	0.9375	0.9524
ZooKeeper	1.0	0.377	0.6512	0.6512	0.6512	0.9945	0.988	0.7321	0.8367	0.7810
OpenStack	1	0.377	0.6512	0.6512	0.6512	1	0.9685	0.9302	0.9302	0.9302
BGL	0.991	0.89	0.7521	0.75833	0.7552	0.991	0.9935	0.9504	0.9583	0.9544
HPC	0.9325	0.91	0.6604	0.7609	0.7071	0.978	0.99	0.7736	0.9318	0.8454
Thunderbird	0.6745	0.355	0.4220	0.4899	0.4534	0.6745	0.935	0.7341	0.8523	0.7888
Windows	0.6915	0.676	0.6364	0.7	0.6667	0.996	0.994	0.8727	0.9796	0.9231
Linux	0.3575	0.196	0.6087	0.5932	0.6009	1	0.938	0.9304	0.9304	0.9304
Mac	0.898	0.4415	0.4583	0.4516	0.4549	0.9525	0.8275	0.6935	0.7191	0.7061
Android	0.8465	0.526	0.5466	0.5301	0.5382	0.95	0.8515	0.9068	0.9419	0.9241
HealthApp	0.9025	0.6185	0.6712	0.6533	0.6622	1	0.8595	0.9041	0.9041	0.9041
Apache	1	0.7155	0.8333	0.8333	0.8333	1	1	1	1	1
OpenSSH	1	0.109	0.2593	0.2593	0.2593	1	1	1	1	1
Proxifier	0.0485	0	0	0	0	1	1	1	1	1
Average	0.8256	0.4623	0.4940	0.5024	0.4974	0.9658	0.9539	0.8828	0.9266	0.9032
Best of others	0.87	0.29	0.27	0.29	0.28	0.83	0.35	0.30	0.33	0.31
	Parsing with Industrial Data (Regex-based)					Parsing with Industrial Data (Best of others)				
Dataset	GA	PA	PTA	RTA	FTA	GA	PA	PTA	RTA	FTA
Huawei	0.9975	0.98	0.9565	0.9778	0.9670	0.3055	0.1	0.1923	0.1111	0.1408

approach. We also have certain ambiguous issues that could be overlooked while editing the Ground Truth files, which could impact the outcomes. Modifying Ground Truth files is a very rigorous and meticulous work, even one space will affect the final result. To address this issue, we had three people to conduct these studies in order to reduce this threat and make sure the process of change was uniform. To look for inconsistencies, four people conducted manual inspections and compared the trial data. We also made the updated Ground Truth files accessible to everyone.

VII. Discussion and Conclusion

Our study showed that our parser performed well on both existing and industrial log data. However, there were limitations to our study that could be addressed in future research. For example, we only focused on word-level classification tasks and did not explore other approaches to log parsing. To address these limitations, potential avenues for research include exploring deep learning techniques, using domain-specific knowledge or ontologies, and developing log parsing methods for different log formats. Additionally, we suggest the need to develop standardized evaluation metrics and datasets to enable direct comparisons between parsers and promote progress in the field.

In this paper, we selected a diverse set of datasets of varying complexity, and source, each with an equal number of log messages to ensure fair and unbiased evaluation results. We evaluated each parser on all datasets and used multiple evaluation metrics to assess parser performance from different perspectives. Although other variables could have influenced our results, we believe our study design and methodology were robust enough to minimize their impact. However, we acknowledge that factors such as the complexity, variety, and quality of logs can still affect parser performance and should be considered in the evaluation process. To address this issue, future studies can take steps to better control for potential confounding variables. This could involve using more standardized datasets or statistical techniques such as randomization or matching to ensure differences in performance are not simply due to differences in datasets. Researchers can also be more transparent in reporting their methodology and results, including details about dataset selection and processing, and any preprocessing steps taken. This will help readers better understand the study's limitations and how the results can be applied in practice.

In conclusion, the empirical study conducted in this paper sheds light on the current state of log parsing research and provides important insights for future directions. The results showed significant variation in parser performance under different evaluation metrics and emphasized the importance of preprocessing in log parsing. The study also revealed that the commonly used LogHub dataset contains labeling errors and a revised Ground Truth was provided after extensive manual efforts. Finally, the results suggest that formulating log parsing as a word-level classification task may be a feasible future direction, as the simple parser based on word-level regular expressions outperformed existing parsers in terms of different metrics on various datasets. A number of conclusions can be summed up.

- Under the current metrics, we produce a wide range of findings. Different metrics behave very differently in evaluating log parsers, and may display very different results under a single parser.
- The analytical results for various datasets are quite disparate.
- Preprocessing can impact the log parsers. The amount and quality of the impact depend on factors like the dataset and regular expressions used.
- Parser technology is not poor. Our regex-based parser can still produce good results using the original Ground Truth across different metrics.
- Errors in the log content and Ground Truth greatly affect the performance of the regex-based parsing. But after revising the Ground Truth, the performance improved a lot. We have successfully posted the latest version on GitHub.
- Our regex-parser can process logs much more efficiently than other parsers. However, regular expressions can not handle all uncertainties, so we may need to consider other options.

Based on our results, we recommend that practitioners pay close attention to preprocessing steps when parsing logs, as these can have a significant impact on parser performance. In particular, we found that removing irrelevant or redundant information from logs prior to parsing can improve accuracy and reduce labeling errors. Additionally, we suggest that practitioners consider using word-level classification tasks as a feasible approach to log parsing, as this method outperformed existing parsers in our experiments. Overall, we believe that our study provides valuable insights into the current state of log parsing research and highlights areas for improvement in automated log analysis. By taking into account the recommendations outlined in our paper, practitioners can improve the accuracy and usefulness of their log parsing methods and ultimately enhance their ability to extract insights from software runtime logs.

VIII. Acknowledgement

This paper was supported by the National Natural Science Foundation of China (No. 62102340).

References

[1] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "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, pp. 654–661.

[2] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *ICWS'17: Proc. of the 24th International Conference on Web Services*, 2017.

[3] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 167–16 710.

[4] W. Meng, Y. Liu, S. Zhang, F. Zaiter, Y. Zhang, Y. Huang, Z. Yu, Y. Zhang, L. Song, M. Zhang *et al.*, "Logclass: Anomalous log identification and classification with partial labels," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1870–1884, 2021.

[5] Z. A. Khan, D. Shin, D. Bianculli, and L. Briand, "Guidelines for assessing the accuracy of log message template identification techniques," in *Proceedings of the 44th International Conference on Software Engineering (ICSE'22)*. ACM, 2022.

[6] H. Dai, H. Li, C. S. Chen, W. Shang, and T.-H. Chen, "Logram: Efficient log parsing using n-gram dictionaries," *IEEE Transactions on Software Engineering*, 2020.

[7] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin, Y. Dang *et al.*, "Uniparser: A unified log parser for heterogeneous log data," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1893–1901.

[8] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *ICSE'19: Companion Proc. of the 41st International Conference on Software Engineering (SEIP)*, 2019.

[9] L. Zhang, X. Xie, K. Xie, Z. Wang, Y. Lu, and Y. Zhang, "An efficient log parsing algorithm based on heuristic rules," in *Advanced Parallel Processing Technologies: 13th International Symposium, APPT 2019, Tianjin, China, August 15–16, 2019, Proceedings 13*. Springer, 2019, pp. 123–134.

[10] W. Meng, Y. Liu, F. Zaiter, S. Zhang, Y. Chen, Y. Zhang, Y. Zhu, E. Wang, R. Zhang, S. Tao *et al.*, "Logparse: Making log parsing adaptive through word classification," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2020, pp. 1–9.

[11] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: a large collection of system log datasets towards automated log analytics," *arXiv preprint arXiv:2008.06448*, 2020.

[12] ——, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 2016, pp. 207–218.

[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] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, "Abstracting execution logs to execution events for enterprise applications," in *QSIC*, no. 181-186, 2008.

[15] 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.

[16] 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.

[17] M. Nagappan and M. A. Vouk, "Abstracting log lines to log event types for mining software system logs," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 2010, pp. 114–117.

[18] 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.

[19] M. Mizutani, "Incremental mining of system log format," in *2013 IEEE International Conference on Services Computing*, 2013, pp. 595–602.

[20] R. Vaarandi and M. Pihelgas, "Logcluster-a data clustering and pattern mining algorithm for event logs," in *2015 11th International conference on network and service management (CNSM)*. IEEE, 2015, pp. 1–7.

[21] K. Shima, "Length matters: Clustering system log messages using length of words," 11 2016.

[22] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *CIKM'16 Proc. of the 25th ACM International Conference on Information and Knowledge Management*, 2016.

[23] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *ICDM'16 Proc. of the 16th International Conference on Data Mining*, 2016.

[24] L. Yu, T. Wu, J. Li, P. Chan, H. Min, and F. Meng, "Documentation based semantic-aware log parsing," *arXiv preprint arXiv:2202.07169*, 2022.

[25] J. Rand and A. Miranskyy, "On automatic parsing of log records," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 2021, pp. 41–45.
[26] W. Liu, X. Liu, X. Di, and B. Cai, "Fastlogsim: a quick log pattern parser scheme based on text similarity," in *International Conference on Knowledge Science, Engineering and Management*. Springer, 2020, pp. 211–219.
[27] L. Bao, N. Busany, D. Lo, and S. Maoz, "Statistical log differencing," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 851–862.
[28] R. Vaarandi, "Mining event logs with slct and loghound," in *NOMS 2008-2008 IEEE Network Operations and Management Symposium*. IEEE, 2008, pp. 1071–1074.
[29] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 683–694.
[30] L. Wang, N. Zhao, J. Chen, P. Li, W. Zhang, and K. Sui, "Root-cause metric location for microservice systems via log anomaly detection," in *2020 IEEE International Conference on Web Services (ICWS)*. IEEE, 2020, pp. 142–150.
[31] S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang, "Log-based abnormal task detection and root cause analysis for spark," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 389–396.
[32] S. Zhang, D. Li, Y. Sun, W. Meng, Y. Zhang, Y. Zhang, Y. Liu, and D. Pei, "Unified anomaly detection for syntactically diverse logs in cloud datacenter," *Journal of Computer Research and Development*, vol. 57, no. 4, p. 778, 2020.
[33] R. Chen, S. Zhang, D. Li, Y. Zhang, F. Guo, W. Meng, D. Pei, Y. Zhang, X. Chen, and Y. Liu, "Logtransfer: Cross-system log anomaly detection for software systems with transfer learning," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 37–47.
[34] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Plelog: semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2021, pp. 230–231.
[35] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," 2022.
[36] S. Lu, X. Wei, B. Rao, B. Tak, L. Wang, and L. Wang, "Ladra: Log-based abnormal task detection and root-cause analysis in big data processing with spark," *Future Generation Computer Systems*, vol. 95, pp. 392–403, 2019.
[37] D. Borthakur, "Hdfs architecture guide," *Hadoop Project Website*, 2008.
[38] A. Hadoop, "Apache hadoop," *URL http://hadoop. apache. org*, 2011.
[39] A. Spark, "Apache spark," *Retrieved January*, vol. 17, no. 1, p. 2018, 2018.
[40] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *DSN'07*, 2007.
[41] U. Kukreja, W. E. Stevenson, and F. E. Ritter, "Rui: Recording user input from interfaces under windows and mac os x," *Behavior Research Methods*, vol. 38, no. 4, pp. 656–659, 2006.
[42] J. M. Hart, *Windows system programming*. Pearson Education, 2010.