2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)

Tools and Benchmarks for Automated Log Parsing

Jieming Zhu[¶], Shilin He[†], Jinyang Liu[‡*], Pinjia He[§], Qi Xie[‖], Zibin Zheng[‡], Michael R. Lyu[†]

[¶]Huawei Noah's Ark Lab, Shenzhen, China
[†]Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong
[‡]School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China
[§]Department of Computer Science, ETH Zurich, Switzerland
[‖]School of Computer Science and Technology, Southwest Minzu University, Chengdu, China

jmzhu@ieee.org, slhe@cse.cuhk.edu.hk, liujy@logpai.com, pinjiahe@gmail.com
qi.xie.swun@gmail.com, zhzibin@mail.sysu.edu.cn, lyu@cse.cuhk.edu.hk

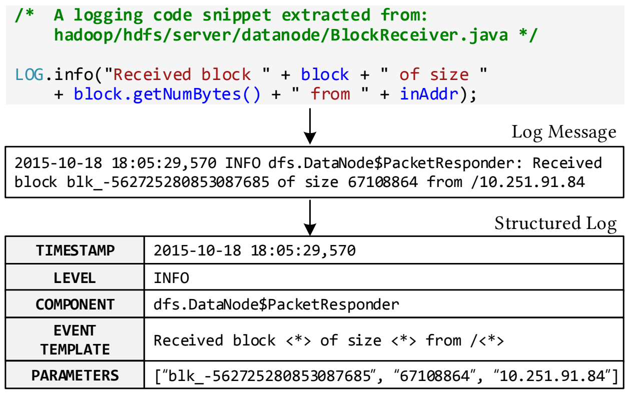
***Abstract*—Logs are imperative in the development and maintenance process of many software systems. They record detailed runtime information that allows developers and support engineers to monitor their systems and dissect anomalous behaviors and errors. The increasing scale and complexity of modern software systems, however, make the volume of logs explodes. In many cases, the traditional way of manual log inspection becomes impractical. Many recent studies, as well as industrial tools, resort to powerful text search and machine learning-based analytics solutions. Due to the unstructured nature of logs, a first crucial step is to parse log messages into structured data for subsequent analysis. In recent years, automated log parsing has been widely studied in both academia and industry, producing a series of log parsers by different techniques. To better understand the characteristics of these log parsers, in this paper, we present a comprehensive evaluation study on automated log parsing and further release the tools and benchmarks for easy reuse. More specifically, we evaluate 13 log parsers on a total of 16 log datasets spanning distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software. We report the benchmarking results in terms of accuracy, robustness, and efficiency, which are of practical importance when deploying automated log parsing in production. We also share the success stories and lessons learned in an industrial application at Huawei. We believe that our work could serve as the basis and provide valuable guidance to future research and deployment of automated log parsing.**

***Index Terms*—Log management, log parsing, log analysis, anomaly detection, AIOps**

I. Introduction

Logs play an important role in the development and maintenance of software systems. It is a common practice to record detailed system runtime information into logs, allowing developers and support engineers to understand system behaviours and track down problems that may arise. The rich information and the pervasiveness of logs enable a wide variety of system management and diagnostic tasks, such as analyzing usage statistics [1], ensuring application security [2], identifying performance anomalies [3], [4], and diagnosing errors and crashes [5], [6].

Despite the tremendous value buried in logs, how to analyze them effectively is still a great challenge [7]. First, modern



```
/*  A logging code snippet extracted from:
    hadoop/hdfs/server/datanode/BlockReceiver.java */

LOG.info("Received block " + block + " of size "
    + block.getNumBytes() + " from " + inAddr);
```

```
2015-10-18 18:05:29,570 INFO dfs.DataNode$PacketResponder: Received
block blk_-562725280853087685 of size 67108864 from /10.251.91.84
```

TIMESTAMP	2015-10-18 18:05:29,570
LEVEL	INFO
COMPONENT	dfs.DataNode$PacketResponder
EVENT TEMPLATE	Received block <*> of size <*> from /<*>
PARAMETERS	["blk_-562725280853087685", "67108864", "10.251.91.84"]

Fig. 1. An Illustrative Example of Log Parsing

software systems routinely generate tons of logs (e.g., about gigabytes of data per hour for a commercial cloud application [8]). The huge volume of logs makes it impractical to manually inspect log messages for key diagnostic information, even provided with search and grep utilities. Second, log messages are inherently unstructured, because developers usually record system events using free text for convenience and flexibility [9]. This further increases the difficulty in automated analysis of log data. Many recent studies (e.g., [10]–[12]), as well as industrial solutions (e.g., Splunk [13], ELK [14], Logentries [15]), have evolved to provide powerful text search and machine learning-based analytics capabilities. To enable such log analysis, the first and foremost step is log parsing [9], a process to parse free-text raw log messages into a stream of structured events.

As the example illustrated in Fig.1, each log message is printed by a logging statement and records a specific system event with its message header and message content. The message header is determined by the logging framework and thus can be relatively easily extracted, such as timestamp, verbosity level (e.g., ERROR/INFO/DEBUG), and component. In contrast, it is often difficult to structurize the free-text message content written by developers, since

[*]Part of the work was done when the author was an intern at Huawei.

it is a composition of `constant` strings and `variable` values. The constant part reveals the event template of a log message and remains the same for every event occurrence. The variable part carries dynamic runtime information (i.e., parameters) of interest, which may vary among different event occurrences. The goal of log parsing is to convert each log message into a specific event template (e.g., "`Received block <*> of size <*> from /<*>`") associated with key parameters (e.g., ["`blk_-562725280853087685`", "`67108864`", "`10.251.91.84`"]). Here, "<*>" denotes the position of each parameter.

The traditional way of log parsing relies on handcrafted regular expressions or grok patterns [16] to extract event templates and key parameters. Although straightforward, manually writing ad-hoc rules to parse a huge volume of logs is really a time-consuming and error-prone pain (e.g., over 76K templates in our Android dataset). Especially, logging code in modern software systems usually update frequently (up to thousands of log statements every month [17]), leading to the inevitable cost of regularly revising these handcrafted parsing rules. To reduce the manual efforts in log parsing, some studies [18], [19] have explored the static analysis techniques to extract event templates from source code directly. While it is a viable approach in some cases, source code is not always accessible in practice (e.g., when using third-party components). Meanwhile, non-trivial efforts are required to build such a static analysis tool for software systems developed across different programming languages.

To achieve the goal of automated log parsing, many data-driven approaches have been proposed from both academia and industry, including frequent pattern mining (SLCT [20], and its extension LogCluster [21]), iterative partitioning (IPLoM [22]), hierarchical clustering (LKE [23]), longest common subsequence computation (Spell [24]), parsing tree (Drain [25]), etc. In contrast to handcrafted rules and source code-based parsing, these approaches are capable of learning patterns from log data and automatically generating common event templates. In our previous work [9], we have conducted an evaluation study of four representative log parsers and made the first step towards reproducible research and open-source tools for automated log parsing. This, to some extent, facilitates some recent developments of tools such as LenMa [26], LogMine [27], Spell [24], Drain [25], and MoLFI [28]. Even more, automated log parsing lately becomes an appealing selling point in some trending log management solutions (e.g., Logentries [15] and Loggly [29]).

In this paper, we present a more comprehensive study on automated log parsing and further publish a full set of tools and benchmarks to researchers and practitioners. In reality, companies are usually reluctant to open their system logs due to confidential issues, leading to the scarcity of real-world log data. With close collaborations with our industrial partners, as well as some pioneer researchers (authors from [10], [18], [30]), we collect a large set of logs (over 77GB in total) produced by 16 different systems spanning distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software. Since the first release of these logs [31], they have been requested by over 150 organizations from both industry and academia.

Meanwhile, the lack of publicly-available tools hinders the adoption of automated log parsing. Therefore, we release an easy-to-use, open-source toolkit[1], with a total of 13 recently-published log parsing methods. We evaluate them thoroughly on 16 different log datasets and report the results in terms of accuracy, robustness, and efficiency. The benchmarking results could help users better understand the characteristics of different log parsers and guide the deployment of automated log parsing in production. We also share the success stories and lessons learned in an industrial application at Huawei. We believe that the availability of tools and benchmarks, as well as the industrial experiences shared in this study, would benefit future research and facilitate wide adoption of automated log parsing in industry.

The remainder of the paper is organized as follows. Section II reviews the state-of-the-art log parsers. Section III reports the benchmarking results. We share our industrial deployment in Section IV, and summarize the related work in Section V. Finally, we conclude the paper in Section VI.

II. Log Parsing

In this section, we present some motivating applications of log parsing, review the characteristics and techniques of existing log parsers, and then describe our tool implementation.

A. Motivating Applications

Log parsing typically serves as the first step towards downstream log analysis tasks. Parsing textual log messages into a structured format enables efficient search, filtering, grouping, counting, and sophisticated mining of logs. To illustrate, we provide a list of sample industrial applications here, which have been widely studied by researchers and practitioners.

- *Usage analysis*. Employing logs for usage analysis is a common task during software development and maintenance. Typical examples include user behaviour analysis (e.g., Twitter [1]), API profiling, log-based metrics counting (e.g., Google Cloud [32]), and workload modeling (e.g., Microsoft [33]). These applications typically require structured events as inputs.
- *Anomaly detection*. Anomaly detection nowadays plays a central role in system monitoring. Logs record detailed execution information and thus serve as a valuable data source to detect abnormal system behaviours. Some recent work has investigated the use of machine learning techniques (e.g., PCA [18], invariant mining [34], and deep learning [10]) for anomaly detection. In such cases, log parsing is a necessary data preprocessing step to train machine learning models.
- *Duplicate issue identification*. In practice, system issues (e.g., disk error, network disconnection) often recur or can be repeatedly reported by different users, leading

[1]https://github.com/logpai/logparser

TABLE I
SUMMARY OF INDUSTRIAL LOG MANAGEMENT TOOLS AND SERVICES

Property	**Splunk**	**VMWare Log Insight**	**Azure Log Analytics**	**ELK**	**Graylog**	**Logentries**	**Loggly**	**Logz.io**	**Sumo Logic**	**Insight finder**
Founded Year	2003	N/A	N/A	2010	2012	2010	2009	2014	2010	2015
Product Type	On-Premises /SaaS	SaaS	SaaS	On-Premises /SaaS	On-Premises	SaaS	SaaS	SaaS	SaaS	On-Premises /SaaS
Automated Log Parsing	✓	✓	✓	✗	✗	✓	✓	✓	✗	✗
Custom Parsing	Regex	N/A	Parse operator	Grok	Regex	Regex	Regex	Grok	Regex	N/A
ML Anaytics	✓	✓	✓	✓	N/A	✓	N/A	✓	✓	✓

TABLE II
SUMMARY OF AUTOMATED LOG PARSING TOOLS

Log Parser	**Year**	**Technique**	**Mode**	**Efficiency**	**Coverage**	**Preprocessing**	**Open Source**	**Industrial Use**
SLCT	2003	Frequent pattern mining	Offline	High	✗	✗	✓	✗
AEL	2008	Heuristics	Offline	High	✓	✓	✗	✓
IPLoM	2012	Iterative partitioning	Offline	High	✓	✗	✗	✗
LKE	2009	Clustering	Offline	Low	✓	✓	✗	✓
LFA	2010	Frequent pattern mining	Offline	High	✓	✗	✗	✗
LogSig	2011	Clustering	Offline	Medium	✓	✗	✗	✗
SHISO	2013	Clustering	Online	High	✓	✗	✗	✗
LogCluster	2015	Frequent pattern mining	Offline	High	✗	✗	✓	✓
LenMa	2016	Clustering	Online	Medium	✓	✗	✓	✗
LogMine	2016	Clustering	Offline	Medium	✓	✓	✗	✓
Spell	2016	Longest common subsequence	Online	High	✓	✗	✗	✗
Drain	2017	Parsing tree	Online	High	✓	✓	✓	✗
MoLFI	2018	Evolutionary algorithms	Offline	Low	✓	✓	✓	✗

to many duplicate issues. It is crucial to automatically identify duplicate issues to reduce the efforts of developers and support engineers. Microsoft has reported some studies [11], [35], [36] on this task, in which structured event data are required.

- *Performance modeling*. Facebook has recently reported a use case [3] to apply logs as a valuable data source to performance modeling, where potential performance improvements can be quickly validated. A prerequisite to this approach is to extract all possible event templates from the logs. The performance model construction takes event sequences as inputs.
- *Failure diagnosis*. Manual failure diagnosis is a time consuming and challenging task since logs are not only of huge volume but also extremely verbose and messy. Some recent progress [4], [37] has been made to automate root cause analysis based on machine learning techniques. Likewise, log parsing is deemed as a prerequisite.

B. Characteristics of Log Parsers

As an important step in log analysis, automated approaches of log parsing have been widely studied, producing an abundance of log parsers ranging from research prototypes to industrial solutions. To gain an overview of existing log parsers, we summarize the key characteristics of them.

1) Industrial Solutions. Table I provides a summary of some industrial log analysis and management tools. With the upsurge of big data, many cloud providers as well as startup companies provide on-premise or software-as-a-service (SaaS) solutions for log management. They enable powerful log search, visualization, and machine learning (ML) analytics capabilities. To illustrate, we list 10 representative products in the market, including both well-established ones (e.g., Splunk [13]) and newly-started ones (e.g., Logz.io [38]). As a key component, automated log parsing has recently risen as a appealing selling point in some products [39]–[41]. Current solutions of automated log parsing, however, are realized with built-in parsing support for common log types, such as Apache and Nginx logs [39]. For other types of logs, they have to rely on users to perform custom parsing with regex scripts, grok patterns [16], or a parsing wizard. Current industrial parsing solutions require deep domain knowledge, and thus fall out of the scope of this study.

2) Research Studies. Table II provides a summary of 13 representative log parsers proposed in the literature, which are the main subjects of our study. These log parsers are all aimed for automated log parsing, but may differ in quality. After reviewing the literature, we list some key characteristics for log parsers that are of practical importance.

Technique. Different log parsers may adopt different log parsing strategies. We categorize them into 7 types of strategies, including frequent pattern mining, clustering, iterative partitioning, longest common subsequence, parsing tree, evolutionary algorithms, and other heuristics. We will present more details of these log parsing methods in Section II-C.

Mode. According to different scenarios of log parsing, log parsers can be categorized to two main modes, i.e., offline and online. Offline log parsers are a type of batch processing and require that all the log data are available before parsing. On the contrary, online log parsers process log messages one by one in a streaming manner, which is often more practical when logs are collected as a stream.

Efficiency. Efficiency is always a major concern for log parsing in practice, considering the large volume of logs. An inefficient log parser can greatly hinder subsequent log analysis tasks that have low latency requirements in cases such as real-time anomaly detection and performance monitoring. In Table II, the efficiency of current tools has been categorized into three levels: high, medium and low.

Coverage. Coverage denotes the capability of a log parser to successfully parse all input log messages. If yes, it is marked as "✓". "✗" indicates that a log parser can only structurize part of the logs. For example, SLCT can extract frequently-occurring event templates by applying frequent pattern mining, but fails to handle rare event templates precisely. A high-quality log parser should be able to process all input log messages, since ignoring any important event may miss the opportunity for anomaly detection and root cause identification.

Preprocessing. Preprocessing is a step to remove some common variable values, such as IP address and numbers, by manually specifying simple regular expressions. The preprocessing step is straightforward, but require some additional manual work. We mark "✓" if a preprocessing step is explicitly specified in a log parsing method, and "✗" otherwise.

Open-source. An open-source log parser can allow researchers and practitioners to easily reuse and further improve existing log parsing methods. This can not only benefit related research but also facilitate wide adoption of automated log parsing. However, current open-source tools for log parsing are still limited. We mark "✓" if an existing log parser is open-source, and "✗" otherwise.

Industrial use. A log parser has more practical value and should be more reliable if it has been deployed in production for industrial use. We mark "✓" if a log parser has been reported on use in an industrial setting, and "✗" otherwise.

C. *Techniques of Log Parsers*

In this work, we have studied a total of 13 log parsers. We briefly summarize the techniques used by these log parsers from the following aspects:

1) Frequent Pattern Mining: A frequent pattern is a set of items that occurs frequently in a data set. Likewise, event templates can be seen as a set of constant tokens that occurs frequently in logs. Therefore, frequent pattern mining is an straightforward approach to automated log parsing. Examples include SLCT [20], LFA [42], and LogCluster [21]. All the three log parsers are offline methods and follow a similar parsing procedure: 1) traversing over the log data by several passes, 2) building frequent itemsets (e.g., tokens, token-position pairs) at each traversal, 3) grouping log messages into several clusters, and 4) extracting event templates from each cluster. SLCT, to our knowledge, is the first work that applies frequent pattern mining to log parsing. Furthermore, LFA considers the token frequency distribution in each log message instead of the whole log data to parse rare log messages. LogCluster is an extension of SCLT, and can be robust to shifts in token positions.

2) Clustering: Event template forms a natural pattern of a group of log messages. From this view, log parsing can be modeled as a clustering problem of log messages. Examples that apply the clustering algorithms for log parsing include 3 offline methods (i.e., LKE [23], LogSig [43], and LogMine [27]) and 2 online methods (i.e., SHISO [44], and LenMa [26]). Specifically, LKE employs the hierarchical clustering algorithm based on weighted edit distances between pairwise log messages. LogSig is a message signature based algorithm to cluster log messages into a predefined number of clusters. LogMine can generate event templates in a hierarchical clustering way, which groups log messages into clusters from bottom to top. SHISO and LenMa are both online methods, which parse logs in a similar streaming manner. For each newly coming log message, the parsers first compute its similarity to representative event templates of existing log clusters. The log message will be added to an existing cluster if it is successfully matched, otherwise a new log cluster will be created. Then, the corresponding event template will be updated accordingly.

3) Heuristics: Different from general text data, log messages have some unique characteristics. As such, some work (i.e., AEL [45], IPLoM [22], Drain [25]) proposes heuristics-based log parsing methods. Specifically, AEL separates log messages into multiple groups by comparing the occurrences between constant tokens and variable tokens. IPLoM employs an iterative partitioning strategy, which partitions log messages into groups by message length, token position and mapping relation. Drain applies a fixed-depth tree structure to represent log messages and extracts common templates efficiently. These heuristics make use of the characteristics of logs and perform quite well in many cases.

4) Others: Some other methods exist. For example, Spell [24] utilizes the longest common subsequence algorithm to parse logs in a stream manner. Recently, Messaoudi et al. [28] propose MoLFI, which models log parsing as a multiple-objective optimization problem and solves it using evolutionary algorithms.

D. *Tool Implementation*

Although automated log parsing has been studied for several years, it is still not a well-received technique in industry. This is largely due to the lack of publicly available tools that are ready for industrial use. For operation engineers who often have limited expertise in machine learning techniques, implementing an automated log parsing tool requires non-trivial efforts. This may exceed the overhead for manually crafting regular expressions. Our work aims to bridge this gap between academia and industry and promote the adoption for automated log parsing. We have implemented an open-source log parsing toolkit, namely logparser, and released a large benchmark set as well. As a part-time project, the implementation of logparser takes over two years and have 11.7K LOC in Python.

TABLE III
SUMMARY OF LOGHUB DATASETS

Dataset	Description	Time Span	Data Size	#Messages	#Templates (total)	#Templates (2k)
Distributed system logs						
HDFS	Hadoop distributed file system log	38.7 hours	1.47 GB	11,175,629	30	14
Hadoop	Hadoop mapreduce job log	N.A.	48.61 MB	394,308	298	114
Spark	Spark job log	N.A.	2.75 GB	33,236,604	456	36
ZooKeeper	ZooKeeper service log	26.7 days	9.95 MB	74,380	95	50
OpenStack	OpenStack software log	N.A.	60.01 MB	207,820	51	43
Supercomputer logs						
BGL	Blue Gene/L supercomputer log	214.7 days	708.76 MB	4,747,963	619	120
HPC	High performance cluster log	N.A.	32.00 MB	433,489	104	46
Thunderbird	Thunderbird supercomputer log	244 days	29.60 GB	211,212,192	4,040	149
Operating system logs						
Windows	Windows event log	226.7 days	26.09 GB	114,608,388	4,833	50
Linux	Linux system log	263.9 days	2.25 MB	25,567	488	118
Mac	Mac OS log	7.0 days	16.09 MB	117,283	2,214	341
Mobile system logs						
Android	Android framework log	N.A.	3.38 GB	30,348,042	76,923	166
HealthApp	Health app log	10.5 days	22.44 MB	253,395	220	75
Server application logs						
Apache	Apache server error log	263.9 days	4.90 MB	56,481	44	6
OpenSSH	OpenSSH server log	28.4 days	70.02 MB	655,146	62	27
Standalone software logs						
Proxifier	Proxifier software log	N.A.	2.42 MB	21,329	9	8

Currently, logparser contains a total of 13 log parsing methods proposed by researchers and practitioners. Among them, five log parsers (i.e., SLCT, LogCluster, LenMa, Drain, MoLFI) are open-source from existing research work. However, they are implemented in different programming languages and have different input/output formats. Examples and documents are also missing or incomplete, making it difficult for a trial. For ease of use, we define a standard and unified input/output interface for different log parsing methods and further wrap up the existing tools into a single Python package. Logparser requires a raw log file with free-text log messages as input, and finally outputs a structured log file and an event template file with aggregated event counts. The outputs can be easily fed into subsequent log mining tasks. Our logparser toolkit can help engineers quickly identify the strengths and weaknesses of different log parsing methods and evaluate their possibility for industrial use cases.

III. EVALUATION

In this section, we evaluate 13 log parsers on 16 benchmark datasets, and report the benchmarking results in terms of accuracy, robustness, and efficiency. They are three key qualities of interest when applying log parsing in production.

- *Accuracy* measures the ability of a log parser in distinguishing constant parts and variable parts. Accuracy is one main focus of existing log parsing studies, because an inaccurate log parser could greatly limit the effectiveness of the downstream log mining tasks [9].
- *Robustness* of a log parser measures the consistency of its accuracy under log datasets of different sizes or from different systems. A robust log parser should perform consistently across different datasets, and thus can be used in the versatile production environment.
- *Efficiency* measures the processing speed of a log parser. We evaluate the efficiency by recording the time that a parser takes to parse a specific dataset. The less time a log parser consumes, the higher efficiency it provides.

A. Experimental Setup

Dataset. Real-world log data are currently scarce in public due to confidential issues, which hinders the research and development of new log analysis techniques. In this work, we have released, on our loghub data repository [31], a large collection of logs from 16 different systems spanning distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software. Table III presents a summary of the datasets. Some of them (e.g., HDFS [18], Hadoop [11], BGL [30]) are production logs released by previous studies, while the others (e.g., Spark, Zookeeper, HealthApp, Android) are collected from real-world systems in our lab. Loghub contains a total of 440 million log messages that amounts to 77 GB in size. To the best of our knowledge, it is the largest collection of log datasets. Wherever possible, the logs are not sanitized, anonymized or modified in any way. They are freely accessible for research purposes. At the time of writing, our loghub datasets have been downloaded over 1000 times by more than 150 organizations from both industry (35%) and academia (65%).

In this work, we use the loghub datasets as benchmarks to evaluate all existing log parsers. The large size and diversity

TABLE IV
ACCURACY OF LOG PARSERS ON DIFFERENT DATASETS

Dataset	SLCT	AEL	IPLoM	LKE	LFA	LogSig	SHISO	LogCluster	LenMa	LogMine	Spell	Drain	MoLFI	Best
HDFS	0.545	**0.998**	**1***	**1***	0.885	0.850	**0.998**	0.546	**0.998**	0.851	**1***	**0.998**	**0.998**	**1**
Hadoop	0.423	0.538	**0.954**	0.670	**0.900**	0.633	0.867	0.563	0.885	0.870	0.778	**0.948**	**0.957***	**0.957**
Spark	0.685	**0.905**	**0.920**	0.634	**0.994***	0.544	**0.906**	0.799	0.884	0.576	**0.905**	**0.920**	0.418	**0.994**
Zookeeper	0.726	**0.921**	**0.962**	0.438	0.839	0.738	0.660	0.732	0.841	0.688	**0.964**	**0.967***	0.839	**0.967**
OpenStack	0.867	0.758	0.871*	0.787	0.200	0.200	0.722	0.696	0.743	0.743	0.764	0.733	0.213	0.871
BGL	0.573	0.758	**0.939**	0.128	0.854	0.227	0.711	0.835	0.69	0.723	0.787	**0.963***	**0.960**	**0.963**
HPC	0.839	**0.903***	0.824	0.574	0.817	0.354	0.325	0.788	0.830	0.784	0.654	0.887	0.824	**0.903**
Thunderb.	0.882	**0.941**	0.663	0.813	0.649	0.694	0.576	0.599	**0.943**	**0.919**	0.844	**0.955***	0.646	**0.955**
Windows	0.697	0.690	0.567	**0.990**	0.588	0.689	0.701	0.713	0.566	**0.993**	**0.989**	**0.997***	0.406	**0.997**
Linux	0.297	0.673	0.672	0.519	0.279	0.169	0.701	0.629	0.701*	0.612	0.605	0.690	0.284	0.701
Mac	0.558	0.764	0.673	0.369	0.599	0.478	0.595	0.604	0.698	0.872*	0.757	0.787	0.636	0.872
Android	0.882	0.682	0.712	**0.909**	0.616	0.548	0.585	0.798	0.880	0.504	**0.919***	**0.911**	0.788	**0.919**
HealthApp	0.331	0.568	0.822*	0.592	0.549	0.235	0.397	0.531	0.174	0.684	0.639	0.780	0.440	0.822
Apache	0.731	**1***	**1***	**1***	**1***	0.582	**1***	0.709	**1***	**1***	**1***	**1***	**1***	**1**
OpenSSH	0.521	0.538	0.802	0.426	0.501	0.373	0.619	0.426	**0.925***	0.431	0.554	0.788	0.500	**0.925**
Proxifier	0.518	0.518	0.515	0.495	0.026	**0.967***	0.517	**0.951**	0.508	0.517	0.527	0.527	0.013	**0.967**
Average	0.637	0.754	0.777	0.563	0.652	0.482	0.669	0.665	0.721	0.694	0.751	0.865*	0.605	N.A.

of loghub datasets can not only measure the accuracy of log parsers but also test the robustness and efficiency of them. To allow easy reproduction of the benchmarking results, we randomly sample 2000 log messages from each dataset and manually label the event templates as ground truth. Specifically, in Table III, "#Templates (2k sample)" indicates the number of event templates in log samples, while "#Templates (total)" shows the total number of event templates generated by a rule-based log parser.

Accuracy Metric. To quantify the effectiveness of automated log parsing, as with [24], we define the parsing accuracy (PA) metric as the ratio of correctly parsed log messages over the total number of log messages. After parsing, each log message has an event template, which in turn corresponds to a group of messages of the same template. A log message is considered correctly parsed if and only if its event template corresponds to the same group of log messages as the ground truth does. For example, if a log sequence [E1, E2, E2] is parsed to [E1, E4, E5], we get PA=1/3, since the 2nd and 3rd messages are not grouped together. In contrast to standard evaluation metrics that are used in previous studies, such as precision, recall, and F1-measure [9], [22], [28], PA is a more rigorous metric. In PA, partially matched events are considered incorrect.

The parameters of all the log parsers are fine tuned through over 10 runs and the best results are reported to avoid bias from randomization. All the experiments were conducted on a server with 32 Intel(R) Xeon(R) 2.60GHz CPUs, 62GB RAM, and Ubuntu 16.04.3 LTS installed.

B. Accuracy of Log Parsers

In this part, we evaluate the accuracy of log parsers. We found that some log parsers (e.g., LKE) cannot handle the original datasets in reasonable time (e.g., even in days). Thus, for fair comparison, the accuracy experiments are conducted on sampled subsets, each containing 2,000 log messages. The log messages are randomly sampled from the original log dataset, yet retains the key properties, such as event redundancy and event variety.

Table IV presents the accuracy results of 13 log parsers evaluated on 16 log datasets. Specifically, each row denotes the parsing accuracy of different log parsers on one dataset, which facilitates comparison among different log parsers. Each column represents the parsing accuracy of one log parser over different datasets, which helps identify its robustness across different types of logs. In particular, we mark accuracy values greater than 0.9 in boldface since they indicate high accuracy in practice. For each dataset, the best accuracy is highlighted with a asterisk "*" and shown in the column "Best". We can observe that most of the datasets are accurately (over 90%) parsed by at least one log parser. Totally, 8 out of 13 log parsers attain the best accuracy on at least two log datasets. Even more, some log parsers can parse the HDFS and Apache datasets with 100% accuracy. This is because HDFS and Apache error logs have relatively simple event templates and are easy to identify. However, several types of logs (e.g., OpenStack, Linux, Mac, HealthApp) still could not be parsed accurately due to their complex structure and abundant event templates (e.g., 341 templates in Mac logs). Therefore, further improvements should be made towards better parsing those complex log data.

To measure the overall effectiveness of log parsers, we compute the average accuracy of each log parser across different datasets, as shown in the last row of Table IV. We can observe that, on average, the most accurate log parser is Drain, which attains high accuracy on 9 out of 16 datasets. The other top ranked log parsers include IPLoM, AEL, and Spell, which achieve high accuracy on 6 datasets. In contrast, the four log parsers that have the lowest average accuracy are LogSig, LFA, MoLFI, and LKE. Therefore, we can briefly conclude that log

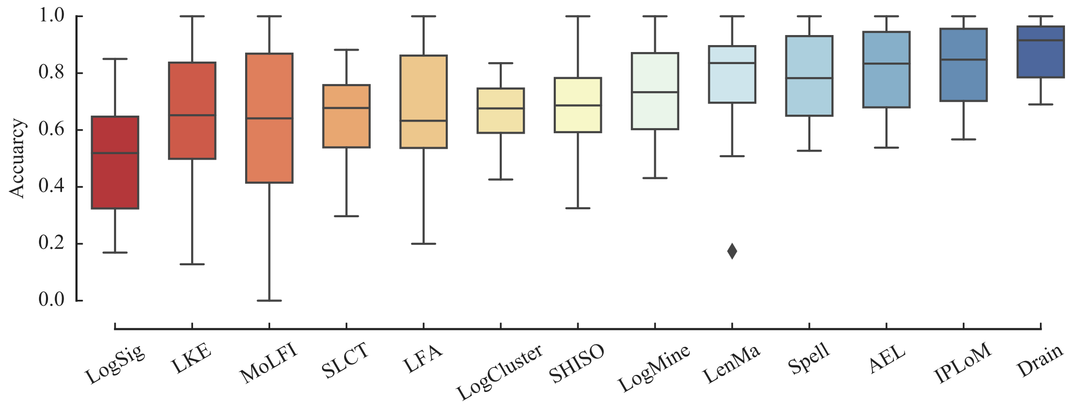


Fig. 2. Accuracy Distribution of Log Parsers across Different Types of Logs

parsers should take full advantage of the inherent structure and characteristics of log messages to achieve good parsing accuracy, instead of directly applying standard algorithms such as clustering and frequent pattern mining.

One may have noticed that the above accuracy results are lower than what have been reported by previous papers (e.g., [25], [27]). The reasons are as follows: 1) We use a more rigorous accuracy metric which rejects partially matched events. 2) For fairness of comparison, we apply the same preprocessing rules (e.g., IP or number replacement) to each log parser, which are much less than those reported before.

C. *Robustness of Log Parsers*

Robustness is crucial to the practical use of a log parser in production environments. In this part, we evaluate the robustness of log parsers from two aspects: 1) robustness across different types of logs and 2) robustness on different volumes of logs.

Figure 2 shows a boxplot that indicates the accuracy distribution of each log parser across the 16 log datasets. For each box, the horizontal lines from bottom to top correspond to the minimum, 25-percentile, median, 75-percentile and maximum accuracy values. The diamond mark denotes an outlier point, since LenMa only has an accuracy of 0.174 on HealthApp logs. From left to right in the figure, the log parsers are arranged in ascending order of the average accuracy shown in Table IV. That is, LogSig has the lowest accuracy and Drain obtains the highest accuracy on average. A good log parser should be able to parse many different types of logs for general use. However, we can observe that, although most log parsers achieve the maximal accuracy over 0.9, they have a large variance over different datasets. There is still no log parser that performs well on all log data. Therefore, we suggest users to try different log parsers on their own logs first. Currently, Drain performs the best among all the 13 log parsers under study. It not only attains the highest accuracy on average, but also shows the smallest variance.

In addition, we evaluate the robustness of log parsers on different volumes of logs. In this experiment, we select six log parsers, i.e., MoLFI, Spell, LenMa, IPLoM, AEL, and Drain. They have achieved high accuracy (over 90%) on more than four log datasets, as shown in Table IV. Meanwhile, MoLFI is the most recently published log parser, and the other five log parsers are ranked in the top in Figure 2. We also choose three large datasets, i.e., HDFS, BGL, and Android. The raw logs have a volume of over 1GB each, and the groundtruth templates are readily available for accuracy computation. HDFS and BGL have also been used as benchmarks datasets in the previous work [22], [24]. For each log dataset, we vary the volume from 300 KB to 1 GB, while fix the parameters of log parsers that were fine tuned on 2k log samples. Specifically, 300KB is roughly the size of each 2k log sample. We truncate the raw log files to obtain samples of other volumes (e.g., 1GB). Figure 3 shows the parsing accuracy results. Note that some lines are incomplete in the figure, because methods like MoLFI and LenMa cannot finish parsing within reasonable time (6 hours in our experiment). A good log parser should be robust to such changes of log volumes. However, we can see that parameters tuned on small log samples cannot fit well to large log data. All the six best performing log parsers have a drop in accuracy or show obvious fluctuations as the log volume increases. The log parsers, except IPLoM, are relatively stable on HDFS data, achieving an accuracy over 80%. Drain and AEL also show relatively stable accuracy on BGL data. However, on Android data, all the parsers have a large degradation on accuracy, because Android logs have quite a large number of event templates and are more complex to parse. Compared to other log parsers, Drain achieves relatively stable accuracy and shows its robustness when changing volumes of logs.

D. *Efficiency of Log Parsers*

Efficiency is an important aspect of log parsers to consider in order to handle log data in large scale. To measure the efficiency of a log parser, we record the running time it needs to finish the entire parsing process. Similar to the setting of the previous experiment, we evaluate six log parsers on three log datasets.

The results are presented in Figure 4. It is obvious that the parsing time increases with the raising of log size on all the three datasets. Drain and IPLoM have better efficiency, which scales linearly with the log size. Both methods can

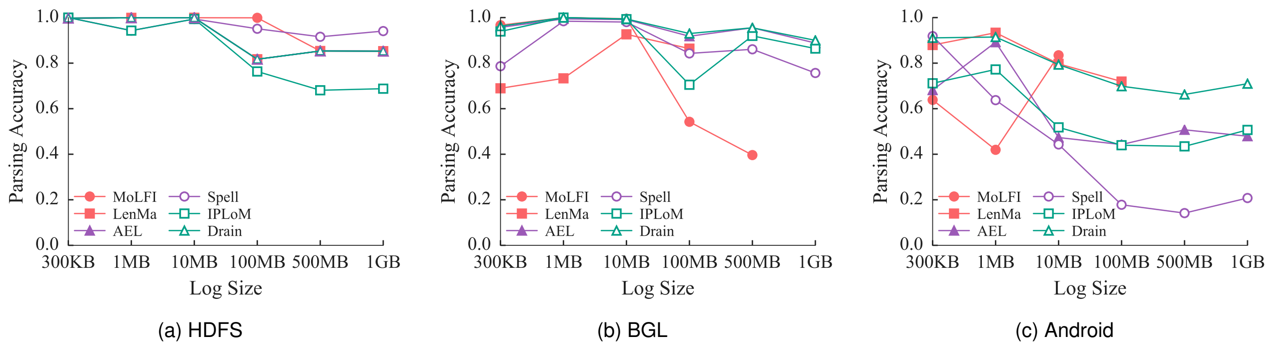


Fig. 3. Accuracy of Log Parsers on Different Volumes of Logs

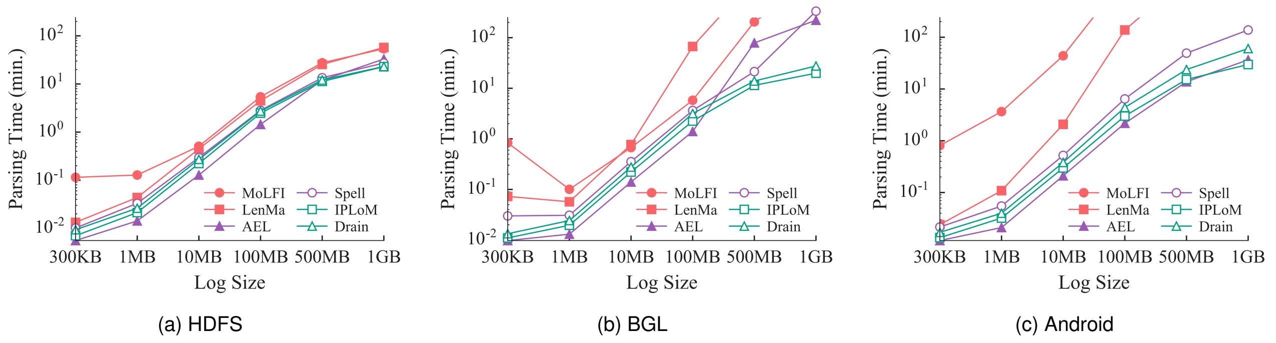


Fig. 4. Efficiency of Log Parsers on Different Volumes of Logs

finish parsing 1GB of logs within tens of minutes. AEL also performs well except on large BGL data. It is because AEL needs to compare with every log message in a bin, yet BGL has a large bin size when the dataset is large. Other log parsers do not scale well with the volume of logs. Especially, LenMa and MoLFI cannot even finish parsing 1GB of BGL data or Android data within 6 hours. The efficiency of a log parser also depends on the type of logs. When the log data is simple and has a limited number of event templates, log parsing is often an efficient process. For instance, HDFS logs contain only 30 event templates, thus all the log parsers can process 1GB of data within an hour. However, the parsing process would become slow for logs with a large number of event templates (e.g., Android).

IV. Industrial Deployment

In this section, we share our experiences of deploying automated log parsing in production at Huawei. System X (anonymized name) is one of the popular products of Huawei. Logs are collected during the whole product lifecycle, from development, testing, beta testing, to online monitoring. They are used as a main data source to failure diagnosis, performance optimization, user profiling, resource allocation, and some other tasks for improving product quality. When the system is still in a small scale, many of these analysis tasks are able to be performed manually. However, after a rapid growth in recent years, System X nowadays produces over terabytes of log data daily. It becomes impractical for engineers to manually inspect logs for diagnostic information, which requires not only nontrivial efforts but also deep knowledge of the logs. In many cases, event statistics and correlations are valuable hints to help engineers make informed decisions.

To reduce the efforts of engineers, a LogKit platform has been built to automate the log analysis process, including log search, rule-based diagnosis, and dashboard reporting of event statistics and correlations. A key feature of this platform is to parse logs into structured data. At first, log parsing was done in an ad-hoc way by writing regular expressions to match the events of interest. However, the parsing rules become unmanageable quickly. First, existing parsing rules cannot cover all types of logs, since it is time-consuming to write the parsing rules one by one. Second, System X is evolving quickly, leading to frequent changes of log structures. Maintenance of such a rule base for log parsing has become a new pain point. As a result, automated log parsing is a high demand.

Success stories. With close collaboration with the product team, we have successfully deployed automated log parsing in production. After detailed comparisons of different log parsers as described in Section III, we choose Drain because of its superiority in accuracy, robustness, and efficiency. In addition, by taking advantage of the characteristics of the logs of System X, we have optimized the Drain approach from the following aspects. *1) Preprocessing*. The logs of System X have over ten thousand event templates as well as a wide range of parameters. As we have done in [9], we apply a simple yet effective preprocessing step to filter common parameters, such as IP, package name, number, and file path. This greatly simplifies the problem for subsequent parsing. Especially, some of the preprocessing scripts are extracted from the original parsing rule base, which is already available. *2) Deduplication*. Many log messages comprise only constant string, with no parameters inside (e.g., "VM terminated."). Recurrences of these log messages result in a large number of duplicate messages in logs. Meanwhile, the preprocessing step produce a lot of duplicate log messages as well (e.g., "Connected to <IP>"), in which common parameters have been removed. We perform deduplication of these duplicate log messages to reduce the data size, which significantly improves the efficiency of log parsing. *3) Partitioning*. The log message header contains two fields: verbosity level and component. In fact, log messages of different levels or components are always printed by different logging statements (e.g., DEBUG vs. INFO). Therefore, it is beneficial to partition log messages into different groups according to the level and component information. This naturally divides the original problem into independent subproblems. *4) Parallelization*. The partitioning of logs can not only narrow down the search space of event templates, but also allow for parallelization. In particular, we extend Drain with Spark and naturally exploit the above log data partitioning for quick parallelization. By now, we have successfully run Drain in production for more than one year, which attains over 90% accuracy in System X. We believe that the above optimizations are general and can be easily extended to other similar systems as well.

Potential improvements. During the industrial deployment of Drain, we have observed some directions that need further improvements. *1) State identification*. State variables are of significant importance in log analysis (e.g., "DB connection ok" vs. "DB connection error"). However, current log parsers cannot distinguish state values from other parameters. *2) Dealing with log messages with variable lengths*. A single logging statement may produce log messages with variable lengths (e.g., when printing a list). Current log parsers are length-sensitive and fail to deal with such cases, thus resulting in degraded accuracy. *3) Automated parameters tuning*. Most of current log parsers apply data-driven approaches to extracting event templates and some model parameters need to be tuned manually. It is desirable to develop a mechanism for automated parameters tuning. We call for research efforts to realize the above potential improvements, which would contribute to better adoption of automated log parsing.

V. Related Work

Log parsing is only a small part of the broad problem of log management. In this section, we review the related work from the aspects of log quality, log parsing, and log analysis.

Log quality. The effectiveness of log analysis is directly determined by the quality of logs. To enhance log quality, recent studies have been focused on providing informative logging guidance or effective logging mechanisms during development. Yuan et al. [46] and Fu et al. [47] report the logging practices in open-source and industrial systems, respectively. Zhu et al. [48] propose LogAdvisor, a classification-based method to make logging suggestions on where to log. Zhao et al. [49] further provide an entropy metric to determine logging points with maximal coverage of a control flow. Yuan et al. [50] design LogEnhancer to enhance existing logging statements with informative variables. Recently, He et al. [51] have conducted an empirical study on the natural language descriptions of logging statements. Ding et al. [52] provide a cost-effective way for dynamic logging with limited overhead.

Log parsing. Log parsing has been widely studied in recent years, which can be categorized into rule-based, source code-based, and data-driven parsing. Most current log management tools support rule-based parsing (e.g., [40], [41]). Some studies [18], [19] make use of static analysis techniques for source code-based parsing. Data-driven log parsing approaches are the main focus of this paper, most of which have been summarized in Section II. More recently, He et al. [53] have studied large-scale log parsing through the parallelization on Spark. Thaler et al. [54] model textual log messages with deep neural networks. Gao et al. [55] apply an optimization algorithm to discover multi-line structures from logs.

Log analysis. Log analysis is a research area that has been studied for decades due to its practical importance. There are an abundance of techniques and applications of log analysis. Typical applications include anomaly detection [12], [18], [23], [56], problem diagnosis [4], [5], runtime verification [57], performance modeling [3], etc. To address the challenges involved in log analysis, many data analytics techniques have been developed. For example, Xu et al. [18] apply the principle component analysis (PCA) to identify anomaly issues. Du et al. [10] investigate the use of deep learning to model event sequences. Lin et al. [11] develop a clustering algorithm to group similar issues. Our work on log parsing serves as the basis to perform such analysis and can greatly reduce the efforts for the subsequent log analysis process.

VI. Conclusion

Log parsing plays an important role in system maintenance, because it serves as the the first step towards automated log analysis. In recent years, many research efforts have been devoted towards automated log parsing. However, there is a lack of publicly available log parsing tools and benchmark datasets. In this paper, we implement a total of 13 log parsing methods and evaluate them on 16 log datasets from different types of software systems. We have opened source our toolkit and released the benchmark datasets to researchers and practice for easy reuse. Moreover, we share our experience of deploying automated log parsing at Huawei. We hope our work, together with the released tools and benchmarks, could facilitate more research on log analysis.

Acknowledgments

This work was supported by the National Key R&D Program of China (2018YFB1004804), the National Natural Science Foundation of China (61722214, 61502401, 61332010), the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (2016ZT06D211), the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14210717 of the General Research Fund). Zibin Zheng is the corresponding author.

References

[1] G. Lee, J. J. Lin, C. Liu, A. Lorek, and D. V. Ryaboy, "The unified logging infrastructure for data analytics at Twitter," *PVLDB*, vol. 5, no. 12, pp. 1771–1780, 2012.

[2] A. Oprea, Z. Li, T. Yen, S. H. Chin, and S. A. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," in *DSN*, 2015, pp. 45–56.

[3] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch, "The mystery machine: End-to-end performance analysis of large-scale internet services," in *OSDI*, 2014, pp. 217–231.

[4] K. Nagaraj, C. E. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *NSDI*, 2012, pp. 353–366.

[5] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy, "Sherlog: error diagnosis by connecting clues from run-time logs," in *ASPLOS*, 2010, pp. 143–154.

[6] X. Xu, L. Zhu, I. Weber, L. Bass, and D. Sun, "POD-Diagnosis: Error diagnosis of sporadic operations on cloud applications," in *DSN*, 2014, pp. 252–263.

[7] A. J. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Commun. ACM*, vol. 55, no. 2, pp. 55–61, 2012.

[8] H. Mi, H. Wang, Y. Zhou, M. R. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1245–1255, 2013.

[9] 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 *DSN*, 2016, pp. 654–661.

[10] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*, 2017, pp. 1285–1298.

[11] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *ICSE*, 2016.

[12] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *ISSRE*, 2016, pp. 207–218.

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

[14] ELK. [Online]. Available: https://www.elastic.co/elk-stack

[15] Logentries. [Online]. Available: https://logentries.com

[16] A beginnerś guide to logstash grok. [Online]. Available: https://logz.io/blog/logstash-grok

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

[18] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *SOSP*, 2009, pp. 117–132.

[19] M. Nagappan, K. Wu, and M. A. Vouk, "Efficiently extracting operational profiles from execution logs using suffix arrays," in *ISSRE*, 2009, pp. 41–50.

[20] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IPOM*, 2003.

[21] R. Vaarandi and M. Pihelgas, "Logcluster - a data clustering and pattern mining algorithm for event logs," in *CNSM*, 2015, pp. 1–7.

[22] A. Makanju, A. Zincir-Heywood, and E. Milios, "Clustering event logs using iterative partitioning," in *KDD*, 2009.

[23] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *ICDM*, 2009, pp. 149–158.

[24] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *ICDM*, 2016, pp. 859–864.

[25] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *ICWS*, 2017, pp. 33–40.

[26] K. Shima, "Length matters: Clustering system log messages using length of words," *arXiv:1611.03213*, 2016.

[27] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "LogMine: fast pattern recognition for log analytics," in *CIKM*, 2016, pp. 1573–1582.

[28] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *ICPC*, 2018.

[29] Loggly: Cloud log management service. [Online]. Available: https://www.loggly.com

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

[31] Loghub: A collection of system log datasets for intelligent log analysis. [Online]. Available: https://github.com/logpai/loghub

[32] Overview of logs-based metrics. [Online]. Available: https://cloud.google.com/logging/docs/logs-based-metrics

[33] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *OSDI*, 2004, pp. 259–272.

[34] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *ATC*, 2010.

[35] R. Ding, Q. Fu, J. G. Lou, Q. Lin, D. Zhang, and T. Xie, "Mining historical issue repositories to heal large-scale online service systems," in *DSN*, 2014, pp. 311–322.

[36] M. Lim, J. Lou, H. Zhang, Q. Fu, A. B. J. Teoh, Q. Lin, R. Ding, and D. Zhang, "Identifying recurrent and unknown performance issues," in *ICDM*, 2014, pp. 320–329.

[37] Automated root cause analysis for spark application failures. [Online]. Available: https://www.oreilly.com/ideas/automated-root-cause-analysis-for-spark-application-failures

[38] Logz.io. [Online]. Available: https://logz.io

[39] New automated log parsing. [Online]. Available: https://blog.rapid7.com/2016/03/03/new-automated-log-parsing

[40] Log parsing - automated, easy to use, and efficient. [Online]. Available: https://logz.io/product/log-parsing

[41] Automated parsing log types. [Online]. Available: https://www.loggly.com/docs/automated-parsing

[42] M. Nagappan and M. A. Vouk, "Abstracting log lines to log event types for mining software system logs," in *MSR*, 2010, pp. 114–117.

[43] L. Tang, T. Li, and C.-S. Perng, "LogSig: Generating system events from raw textual logs," in *CIKM*, 2011, pp. 785–794.

[44] M. Mizutani, "Incremental mining of system log format," in *SCC*, 2013, pp. 595–602.

[45] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, "Abstracting execution logs to execution events for enterprise applications," in *QSIC*, 2008, pp. 181–186.

[46] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in *ICSE*, 2012, pp. 102–112.

[47] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," in *ICSE*, 2014, pp. 24–33.

[48] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *ICSE*, vol. 1, 2015, pp. 415–425.

[49] X. Zhao, K. Rodrigues, Y. Luo, M. Stumm, D. Yuan, and Y. Zhou, "Log20: Fully automated optimal placement of log printing statements under specified overhead threshold," in *SOSP*, 2017, pp. 565–581.

[50] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving software diagnosability via log enhancement," in *ASPLOS*, 2011, pp. 3–14.

[51] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *ASE*, 2018, pp. 178–189.

[52] R. Ding, H. Zhou, J. Lou, H. Zhang, Q. Lin, Q. Fu, D. Zhang, and T. Xie, "Log2: A cost-aware logging mechanism for performance diagnosis," in *ATC*, 2015.

[53] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Trans. Dependable Sec. Comput. (TDSC)*, vol. 15, no. 6, pp. 931–944, 2018.

[54] M. P. Stefan Thaler, Vlado Menkonvski, "Towards a neural language model for signature extraction from forensic logs," in *ISDFS*, 2017.

[55] Y. Gao, S. Huang, and A. G. Parameswaran, "Navigating the data lake with DATAMARAN: automatically extracting structure from log datasets," in *SIGMOD*, 2018, pp. 943–958.

[56] S. He, Q. Lin, J. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *FSE*, 2018, pp. 60–70.

[57] 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*, 2013, pp. 402–411.