Analysing the current log parsing techniques in large-scale software systems

Stefan Petrescu
TU Delft
Delft, Netherlands
petrescu-1@student.tudelft.nl

Abstract

Due to the complexity of nowadays software systems, the amount of logs generated is significantly high. As a result, it has become infeasible for humans to manually investigate this amount of information in a reasonable amount of time. Consequently, efficiently and effectively utilizing logs requires automating the process of log analysis. By using log mining techniques, automated log analysis can tackle arbitrary system log sizes, thus complementing traditional log analysis methods. However, although log mining techniques have an unquestionable utility, their effectiveness depends on the output quality of a process known as 'log parsing'. Thus, as this is of utmost importance, we examined the state-of-the-art log parsing approaches. For this, we conducted a thorough literature survey in which 34 log parsing methods were successfully identified. Moreover, we evaluated the scalability and accuracy for 12 of these. Our findings indicate that, out of the 12, only 2 are realistically scalable, namely Drain and Spell (these two can handle log sizes of 1M lines in a matter of minutes using 1.3GiB of memory). In regards to their accuracy, all parsers achieve over 90% accuracy for 2k log size datasets.

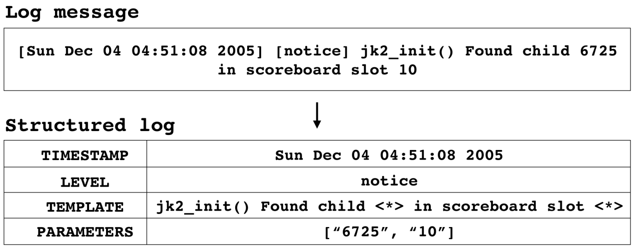
Keywords: log parsing, log abstraction, log signature extraction

1 Introduction

For large-scale software systems, logs record runtime information in order to provide an audit trail that can be used for monitoring, understanding the activity, or diagnosing problems [27]. In practice, logs can be used in various domains such as the financial sector, education, healthcare, defense, etc.

Logs are of utmost importance because they play an essential role in the maintenance of large-scale software systems. For instance, in case of any incidents, system administrators can follow the logs and conduct investigations. Furthermore, in case of errors or anomalies, log messages can facilitate the process of generating alerts.

Due to the complexity of nowadays software systems, log size is significantly high. For example, some systems can produce 30-50 gigabytes of logs per hour [19]. As a result of this, traditional log analysis approaches are no longer suitable [29]. Thus, it has become infeasible for humans to manually investigate this amount of information in a reasonable amount of time. In order to realistically utilize system logs, automated log analysis techniques are required. These can be used for various tasks, such as anomaly detection, problem diagnosis, security assurance etc. [29].



```
[Sun Dec 04 04:51:08 2005] [notice] jk2_init() Found child 6725
                    in scoreboard slot 10
```

TIMESTAMP	Sun Dec 04 04:51:08 2005
LEVEL	notice
TEMPLATE	jk2_init() Found child <*> in scoreboard slot <*>
PARAMETERS	[“6725”, “10”]

Figure 1. Log parsing example; a raw log message gets transformed into structured information. In other words, the main goal of running a log parser is to distinguish between constants and variables [11].

In their foremost step, the vast majority of automated log analysis methods require logs to undergo a specific type of transformation. In the literature, this process is known as 'log parsing'. However, this term is sometimes used interchangeably with 'message signature extraction', 'event template extraction', 'log key extraction', 'message type extraction', 'log abstraction'. During this step, the log parser distinguishes between the constants and the variables, abstracting away from the raw log messages. Based on the specifics of the log dataset, conducting this type of transformation can be problematic, as it might be desirable to account for more than just accuracy (scalability can also be a very important aspect). Nevertheless, this step is of utmost importance as it creates usable input for downstream data mining tasks. An example of how log parsing transforms a raw log message is illustrated in Figure 1.

As the performance of automated log analysis techniques can be directly influenced by the output quality of the log parsing methods [8], it is crucial to find which of these are most appropriate. Hence, this study focuses on the state-of-the-art (SOTA) log parsing approaches in terms of their accuracy and scalability. More specifically, we tackle the following research questions:

RQ1: How has log parsing been approached so far?

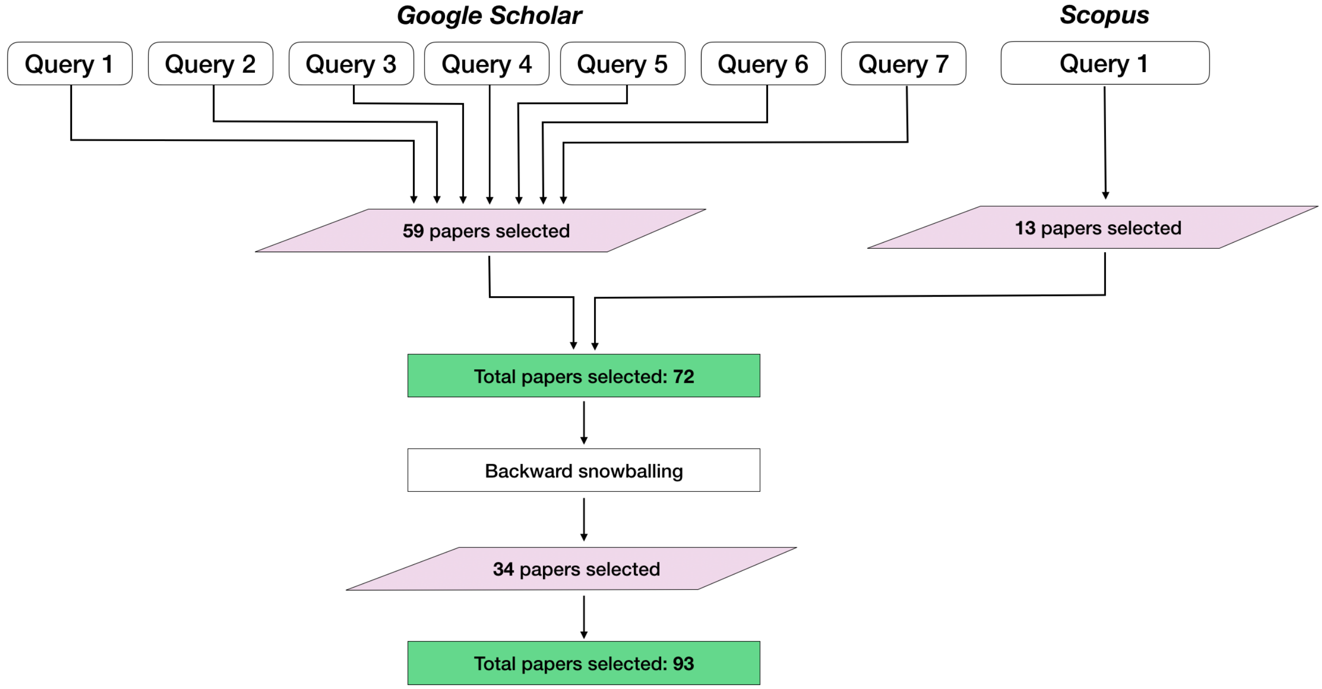


Figure 2. Selection process for the papers that were selected for further investigation.

> **RQ2:** How do log parsing methods perform in terms of their scalability?
> **RQ3:** How do log parsing methods perform in terms of their accuracy?

By answering these questions, we hope to bring more structure and clarity in order to guide future researchers in tackling log parsing.

In regards to the paper's outline, the Method section presents the search methodology. The Results section contains the answers to the research questions. Last but not least, we conclude with the Discussion & Conclusion section.

2 Method

Inspired by the methodology proposed in [15], we conducted the search for relevant papers. In Figure 2, an overview of the paper selection process can be found. Our goal here was to find all possible papers that contained information about log parsing. We considered a paper to be relevant (and thus, selected) either if, it proposed a log parsing method, had a log parser implemented in its pipeline, or had references to other log parsers / log parsing methods. This was decided by reading the title, abstract and, in case of uncertainty, by quickly reading the paper. Furthermore, we have to mention that we only selected papers that were about software logs (logfiles) and not (mathematical) logarithms. Lastly, we considered papers to be relevant only if they tackled software systems.

The selection process started by querying two well-known databases, more specifically Google Scholar[1] and Scopus[2]. For the former, we used 7 queries[3], and for the latter, we used 1 query[3]. For Google Scholar, we consulted the first 10 pages of results (first 100 results, any time, sort by relevance, any type). For Scopus, we consulted all of the 392 returned results. Because we know that Google Scholar dynamically adapts its results based on the users' profiles, we tried to reduce unclarities as much as possible by thoroughly documenting the search and selection process. Thus, both the queries and the selected papers can be found in List of selected papers. Additional information concerning the search method can be found in the "Method" section of the experiment's documentation.

From a total of 93 selected papers, we investigated which of them actually proposed log parsing approaches. After this filtering process, we ended up with 34 papers, that were split in two main categories, namely online and offline approaches (more details about this terminology can be found in the next section). This process can be visualised in in Figure 3.

[1] https://scholar.google.com
[2] https://www.scopus.com
[3] Check List of selected papers

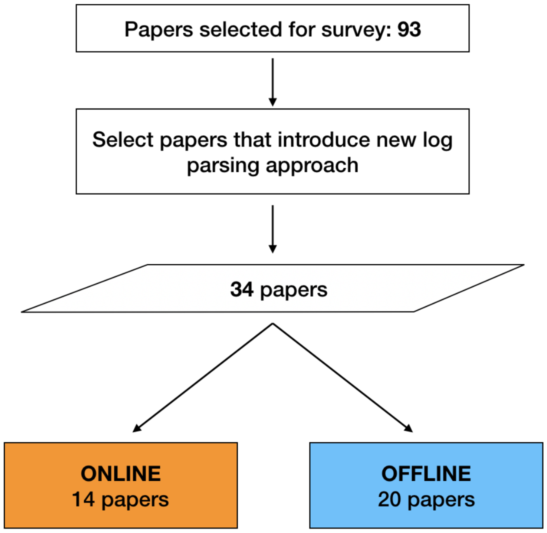

Papers selected for survey: 93
Select papers that introduce new log parsing approach
34 papers
ONLINE
14 papers
OFFLINE
20 papers

Figure 3. Final selection process in order to consider papers that only introduced log parsing approaches.

3 Results

In this section we first briefly present each log parsing approach found and mention its main contribution(s) and biggest disadvantage(s). Furthermore, two representations of the interconnection between the methods are visualised. Each subsection tackles one corresponding research question (mentioned in the Introduction).

3.1 Log parsing approaches

Firstly, inspired by the authors of [11], we categorize log parsing approaches in two main branches, namely offline and online log parsing. For the former, what this means is that, in order for log templates to be generated, all log data needs to be available beforehand. Consequently, getting new event templates means re-running the process (log parsing) periodically (in batches). For the latter, log data is parsed in a streaming manner, making its integration seamless for downstream tasks. As a general overview, Table 1 contains a list of the selected offline log parsing approaches; Table 2 contains a list of the selected online log parsers.

3.1.1 Offline log parsing approaches. Figure 4 presents a visualisation main clusters in which methods can be categorized in, and the interconnection between these. We can see that most of the authors are taking into account other methods when evaluating their own. However, some methods, such as LKE do not evaluate their approach against others (which obviously has some drawbacks). SLCT [24] is one of the first papers published on automated log analysis. These frequent words were utilized in the second pass to find out associated frequent words. Finally, for a log message, if it contained a pattern of associated frequent words, these words would be regarded as constants and employed to generate event templates. Otherwise, the log message would be placed into an outlier cluster without matched event templates [11].

Table 1. Overview of all offline log parsing approaches present in literature.

Year	Method used
2003	SLCT [24]
2008	AEL [14]
2009	LKE [6]
2010	LFA [21]
2011	LogSig [23]
2012	IPLoM [16]
2015	LogCluster [25]
2016	LogMine [7]
2018	POP [8]
2018	MoLFI [18]
2020	LPV [28]

AEL [14] employed a list of specialized heuristic rules. For example, for all the pairs like “word=value,” AEL regarded the “value” as a variable and replaced it with a “$v” symbol.The authors of this paper, amongst other things, evaluate their approach against SLCT.

Another offline log parsing approach is LKE [6]. LKE adopted a hierarchical clustering algorithm with a customized weighted edit distance metric. Additionally, the clusters were further partitioned by heuristic rules. For LFA [21], the adopted a similar strategy as [24]. Differently, LFA could cover all the log messages (which is not the case for SLCT, which can produce outliers that do not belong to any event templates).

LogSig [23], was a more recent clustering-based parser than LKE. Instead of directly clustering log messages, LogSig transformed each log message into a set of word pairs and clustered logs based on the corresponding pairs.

IPLoM [16], contained three steps and partitioned log messages into groups in a hierarchical manner. (1) Partition by log message length. (2) Partition by token position. The position containing the least number of unique words is “token position.” Partitioning was conducted according to the words in the token position. (3) Partition by mapping. Mapping relationships were searched between the set of unique tokens in two token positions, which were selected using a heuristic criterion [11].

LogCluster [25], is similar to SLCT. Differently, LogCluster allowed variable length of parameters in between via a clustering algorithm. Thus, compared with SLCT, LogCluster is better at handling log messages of which the parameter

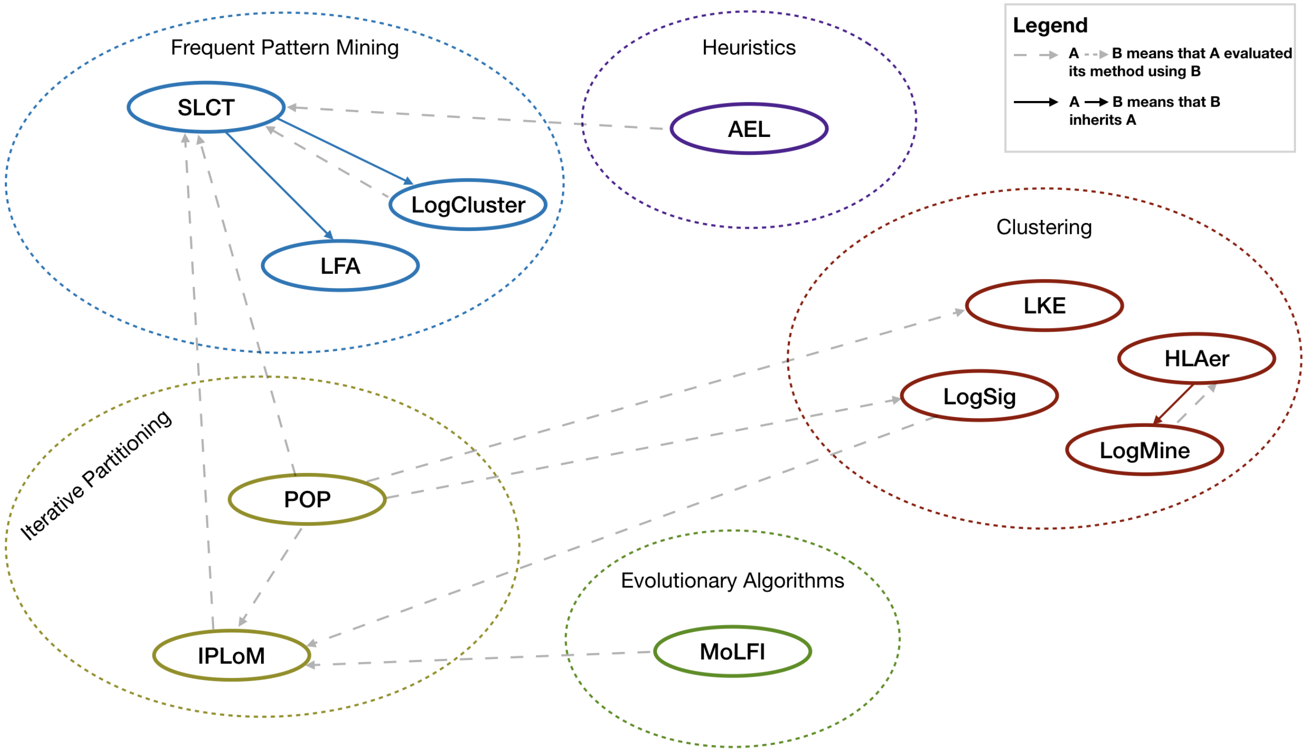


Figure 4. Graphical representation of how the offline methods are clustered together based on their algorithmic approach.

length is flexible. For example, “Download Facebook and install” and “Download Whats App and install” have the same event template “Download <*> and install,” while the length of the parameter (i.e., an app name) is flexible [11]. LogMine [7] adopted an agglomerative clustering algorithm. It was implemented in map-reduce framework for better efficiency [7].

POP [8] is a parallel log parser that utilizes distributed computing to accelerate the parsing of large-scale software logs. POP can parse 200 million HDFS log messages in 7 min, while most of the parsers (e.g., LogSig) fail to terminate in reasonable time. The authors of MoLFI [18] formulate log parsing as a multi-objective optimization problem and propose an evolutionary algorithm-based approach. Specifically, MoLFI employs the Non-dominated Sorting Genetic Algorithm II [3] to search for a Pareto optimal set of event tem- plates. Compared with other log parsers, the strength of MoLFI is that it requires little parameter. Last but not least, for LPV [28], the authors use a clustering-based approach. Instead of grouping raw logs, they cluster vector representations of logs and extract the corresponding templates.

3.1.2 Online log parsing approaches. Figure 5 presents a visualisation main clusters in which the online methods can be categorized in, and the interconnection between these. We can see that authors also evaluate against methods which are in a different mode (offline).

Table 2. Overview of all online log parsing approaches present in literature.

Year	Paper
2013	SHISO [20]
2016	LenMa [22]
2017	Drain [10]
2018	Drain [9]
2019	Spell [4]
2020	Logram [2]
2020	Paddy [13]
2020	LogParse [17]
2020	LPV [28]
2020	OLMPT [26]
2021	Prefix-Graph [1]

SHISO [20], is the first online log parsing approach. SHISO used a tree-form structure to guide the parsing process, where each node was correlated with a log group and an event template. The number of children nodes in all the layers were the same and were manually configured beforehand. During the parsing process, SHISO traversed the tree to find the most suitable log group by comparing the log message and the event templates in the corresponding log groups.

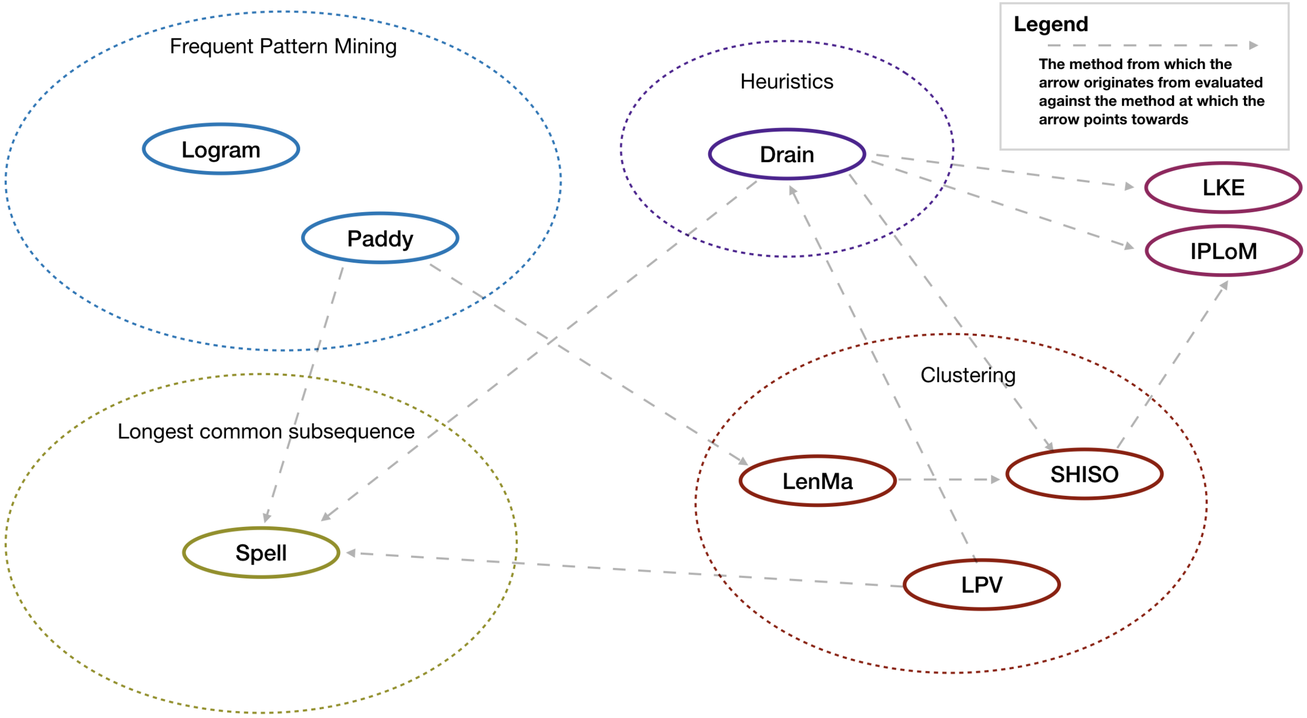

Frequent Pattern Mining
Logram
Paddy
Heuristics
Drain
Legend
The method from which the arrow originates from evaluated against the method at which the arrow points towards
LKE
IPLoM
Clustering
LenMa
SHISO
LPV
Longest common subsequence
Spell

Figure 5. Graphical representation of how the online methods are clustered together based on their algorithmic approach.

SHISO is sensitive to path explosion and thus its efficiency is often unsatisfactory [11].

LenMa [22] is similar to SHISO. LenMa encodes each log message into a length vector, where each dimension records the number of characters of a token. For example, "Receive a file." would be vectorized as [7, 1, 5]. During parsing, LenMa would compare the length vectors of the log messages.

Drain [10] maintained log groups via the leaf nodes in the tree. The internal nodes of the tree embedded different heuristic rules. The extended version of Drain [9] was based on a directed acyclic graph that allowed log group online merging. In addition, it provided the first automated parameter tuning mechanism for log parsing.

Similar to SHISO and LenMa, Spell [4] maintained a list of log groups. To accelerate the parsing process, Spell utilized specialized data structures: prefix tree and inverted index. In addition, Spell provided a parallel implementation [11].

Another method that uses frequent pattern mining is Logram [2]. Different from the existing approaches that count frequent tokens, Logram considered frequent n-gram. The core insight of Logram is: frequent n-gram are more likely to be constants. Note that Logram assumed developers had some log messages on hand to construct the dictionary [11].

Another interesting approach is Paddy [13], in which the authors use a dynamic dictionary in order to build an inverted index. With the help of this, template candidates can be searched efficiently with a high rate of recall.

Last but not least, a very interesting approach can be found in LogParse [17]. Here, the authors propose an adaptive log parsing framework, which can handle software/firmware upgrades in software systems.

3.2 Scalability of log parsing approaches

In order to test scalability, we measured the runtime and the memory consumption of each of the methods. Using the source code provided by the authors of [12], we refactored the algorithms in order to measure both metrics. In order to average the results, we ran each method 10 times, for each dataset (and for its respective size). For the experiments we used a dual socket AMD Epyc2 machine with 64 cores in total (with a dual Nvidia RTX 2080Ti graphics card).

3.2.1 Constructing the datasets. Using the log data provided by the authors of [12], we created 6 different datasets. We selected datasets that had at least 300k log lines, namely Android, BGL, OpenSSH, HDFS, Thunderbird, and Windows. The way in which we constructed the datasets was to select the first k lines from the original data (for example, if a dataset had 330k log lines in total, we selected only the first 300k log lines; if a dataset had 2.5 million log lines in total, we selected only the first million log lines). In order to see how the methods scale, the datasets were split in a number of different sizes, namely 1k, 2k, 4k, 10k, 20k, 50k, 100k, 200k, 300k, 500k, 1M log lines. A visualisation of the constructed datasets

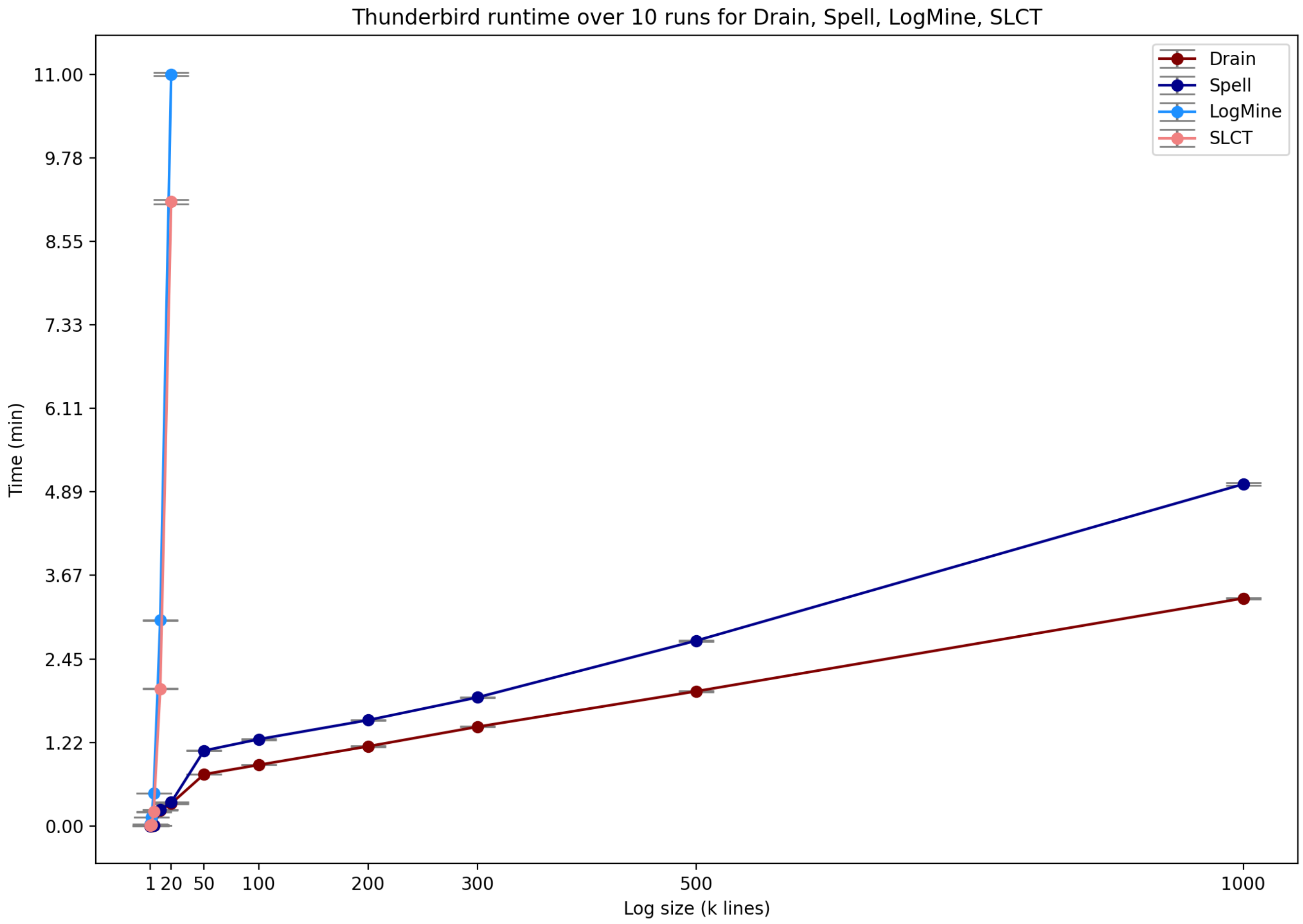


Figure 6. Runtime measurements using the Thunderbird dataset for Drain, Spell, LogMine, and SLCT.

sizes can be found in Figure 7. The scripts used to generate the datasets can be found in this paper's GitHub repository[4].

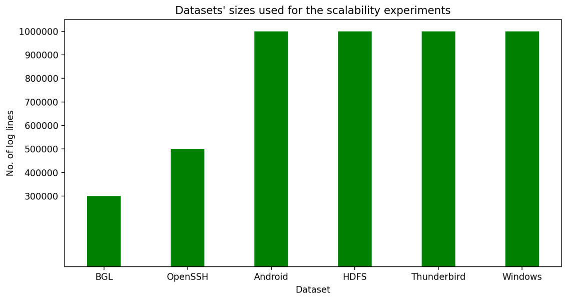


Figure 7. Datasets constructed and their respective sizes. For BGL, the dataset was not big enough to have a 500k or a 1M split. For OpenSSH, the dataset was not big enough to have a 1M split.

3.2.1 Scalability experiments. Unfortunately, not all methods (found in the literature) were able to be tested (either due to the lack of source code, or other difficulties). However, the ones that were able to be tested can be found below (also containing information about the used datasets, and their respective sizes):

1. *Drain*; Android (1k up to 20k); BGL (1k up to 300k); HDFS (1k up to 1M); OpenSSH (1k up to 500k); Thunderbird (1k up to 1M); Windows (1k up to 1M).
2. *Spell*; BGL (1k up to 300k); HDFS (1k up to 1M); OpenSSH (1k up to 500k); Thunderbird (1k up to 1M); Windows (1k up to 20k).
3. *LogMine*; Android (1k up to 20k); BGL (1k up to 20k); HDFS (1k up to 20k); Thunderbird (1k up to 20k);
4. *SLCT*; HDFS (1k up to 1M); Thunderbird (1k up to 20k); Windows (1k up to 10k);
5. *IPLoM*; BGL (1k up to 300k); HDFS (1k up to 1M); OpenSSH (1k up to 500k); Thunderbird (1k up to 1M); Windows (1k up to 1M).

[4] https://github.com/spetrescu/literature-survey-log-parsing

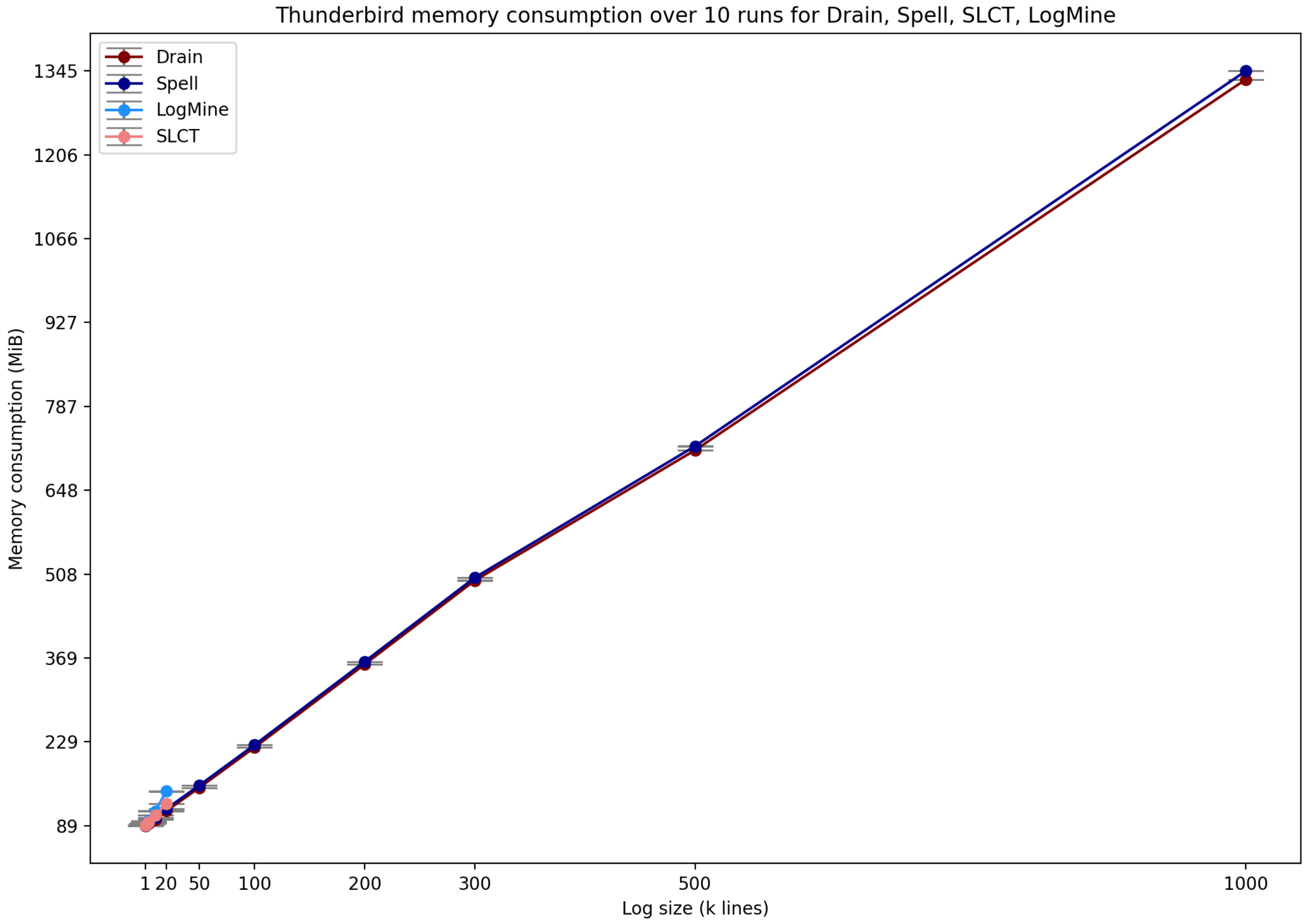


Figure 8. Memory consumption measurements using the Thunderbird dataset for Drain, Spell, LogMine, and SLCT.

6. *AEL*; BGL (1k up to 300k); HDFS (1k up to 500k); OpenSSH (1k up to 500k); Thunderbird (1k up to 500k); Windows (1k up to 500k).
7. *Lenma*; Android (1k up to 200k); BGL (1k up to 300k); HDFS (1k up to 1M); OpenSSH (1k up to 500k); Thunderbird (1k up to 1M); Windows (1k up to 1M);
8. *MoLFI*; BGL (1k up to 50k); HDFS (1k up to 100k); OpenSSH (1k up to 50k); Thunderbird (1k up to 50k); Windows (1k up to 50k).
9. *SHISO*; BGL (1k up to 300k); HDFS (1k up to 300k); OpenSSH (1k up to 300k); Thunderbird (1k up to 300k); Windows (1k up to 300k).
10. *LogCluster*; BGL (1k up to 300k); HDFS (1k up to 1M); OpenSSH (1k up to 500k); Thunderbird (1k up to 1M); Windows (1k up to 1M).
11. *LogSig*; OpenSSH (1k up to 200k); Thunderbird (1k up to 200k); Windows (1k up to 200k).
12. *LKE*; HDFS (1k up to 4k).

A visualisation of the above list can be found in Figure 9, in Figure 10, and in Figure 11 respectively. In [5], the authors claim that SLCT, POP, LogMine, Spell, Drain are scalable. Also, they claim that IPLoM and HLAer are potentially scalable. Thus, we prioritized running the scalability experiments for these first. However, the code for POP and HLAer was not available. Consequently, we were unable to run any experiments for these two. We have to mention that most of the algorithms had problems parsing the Android dataset, which in turn made comparisons between methods impossible. On the other hand, there were no issues running the experiments for the rest of the datasets (and comparisons were thus feasible). Last but not least, we have to mention that the memory consumption measurements are of the entire Python process (not of a function call).

Figure 6 contains a visualisation of the different runtime measurements for the methods claimed to be scalable, namely Drain, Spell, LogMine, and SLCT. We observe a significant difference between Drain & Spell and LogMine & SLCT. The last two require almost 10 minutes to parse 20k logs, compared to the first two, which require rougly 3 and 5 minutes

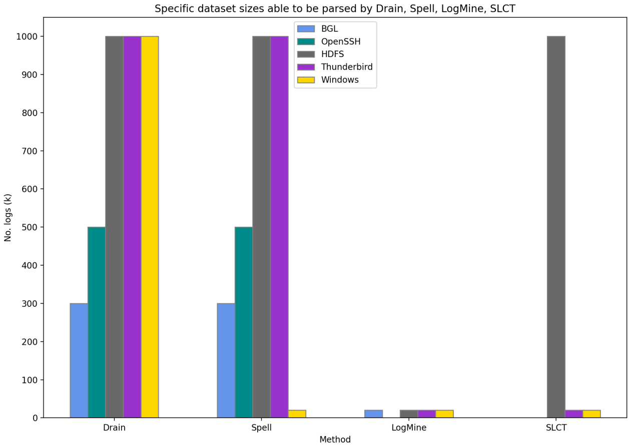


Figure 9. Dataset sizes used for the scalability experiments for Drain, Spell LogMine, and SLCT. If a method parsed a dataset of size N, all smaller splits were also parsed (for example if Drain parsed the 1M Windows dataset, it also parsed 1k, 2k, 4k, 10k, 20k, 50k, 100k, 200k, 300k, 500k log lines during the experiments.

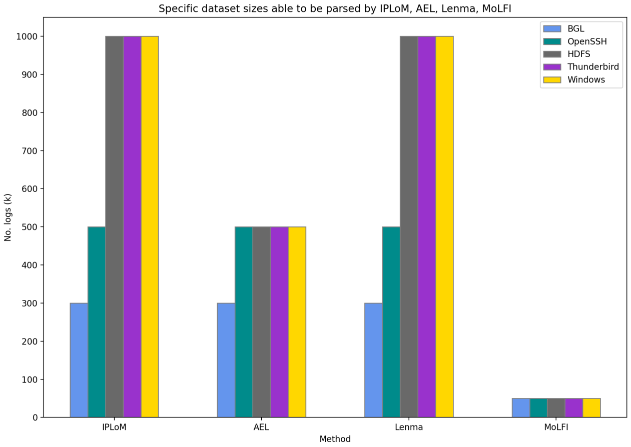


Figure 10. Dataset sizes used for the scalability experiments for IPLoM, AEL, Lenma, and MoLFI. If a method parsed a dataset of size N, all smaller splits were also parsed (for example if IPLoM parsed the 1M Windows dataset, it also parsed 1k, 2k, 4k, 10k, 20k, 50k, 100k, 200k, 300k, 500k log lines during the experiments.

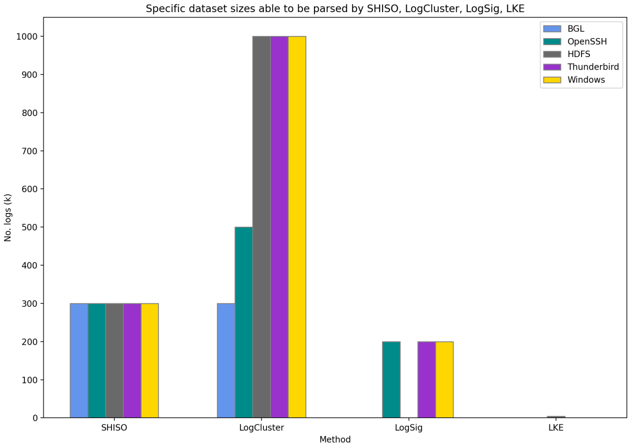


Figure 11. Dataset sizes used for the scalability experiments for SHISO, LogCluster, LogSig, and LKE. If a method parsed a dataset of size N, all smaller splits were also parsed (for example if IPLoM parsed the 1M Windows dataset, it also parsed 1k, 2k, 4k, 10k, 20k, 50k, 100k, 200k, 300k, 500k log lines during the experiments.

to parse 1M logs. A visualisation of the memory consumption for these 4 methods can be seen in Figure 8. We observe that the memory difference is not that significant, compared to the previous (runtime) scenario. Furthermore, we can see how the methods perform on the Windows dataset. For this, a visualisation of the runtime measurements can be found in Figure 12. Also, a visualisation of the memory consumption can be found in Figure 13. For the rest of the methods we created similar visualisations. These can be found in Experiments visualisations.

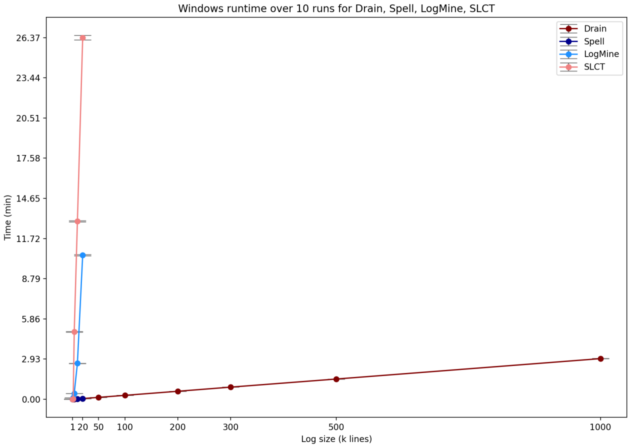


Figure 12. Runtime measurements using the Windows dataset for Drain, Spell, LogMine, and SLCT.

3.3 Accuracy of log parsing approaches

In order to test the methods' accuracy, we reproduced the results[5] of the experiments performed by the authors of [8]. Overall, all methods have an accuracy higher than 90%. They are able to generate templates and are quite robust to various datasets. However, there are some drawbacks, because the datasets only contain 2k logs. As of future improvements, a

[5] These can be found at accuracy results

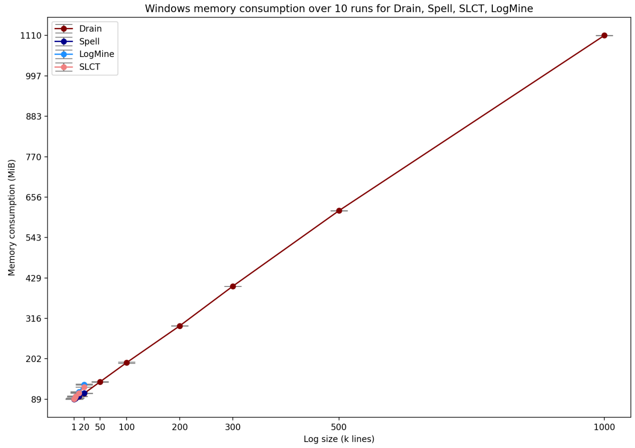


Figure 13. Memory consumption measurements using the Windows dataset for Drain, Spell, LogMine, and SLCT.

new approach for the accuracy experiments could be considered, either by augmenting the already existing datasets, or by constructing new labelled datasets.

4 Discussion & Conclusion

In this paper, SOTA log parsing methods have been analyzed. The aim of this survey was directed towards three aspects, namely mapping out SOTA log parsing approaches, and evaluating the scalability and accuracy of these methods. Regarding the first aspect, log parsing approaches are divided in 2 main categories, offline and online approaches. For offline log parsing approaches, 14 were successfully identified, whereas, for online log parsing approaches, 20 approaches were successfully identified. The answer to the second research question is that it seems like there are only a few scalable methods (out of the ones that were available to be tested) namely, Drain, Spell (these can run their methods in a matter of minutes, and seem robost to various log formats). IPLoM also seemed promising, however it memory consumption is significantly higher than the one used by Drain and Spell. In regards to the third reasearch question, all methods have an accuracy of over 90% on all datasets. However, as a future reasearch direction, it would be wise to test this on larger datasets, as 2k log lines might not be enough in order to draw relevant conclusions.

References

[1] Guojun Chu, Jingyu Wang, Qi Qi, Haifeng Sun, Shimin Tao, and Jianxin Liao. 2021. Prefix-Graph: A Versatile Log Parsing Approach Merging Prefix Tree with Probabilistic Graph. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 2411–2422. https://doi.org/10.1109/ICDE51399.2021.00274

[2] Hetong Dai, Heng Li, Weiyi Shang, Tse-Hsun Chen, and Che-Shao Chen. 2020. Logram: Efficient Log Parsing Using n-Gram Dictionaries. arXiv:2001.03038 [cs.SE]

[3] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. 2000. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In *Parallel Problem Solving from Nature PPSN VI*, Marc Schoenauer, Kalyanmoy Deb, Günther Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 849–858.

[4] Min Du and Feifei Li. 2019. Spell: Online Streaming Parsing of Large Unstructured System Logs. *IEEE Transactions on Knowledge and Data Engineering* 31, 11 (2019), 2213–2227. https://doi.org/10.1109/TKDE.2018.2875442

[5] Diana El-Masri, Fabio Petrillo, Yann-Gaël Guéhéneuc, Abdelwahab Hamou-Lhadj, and Anas Bouziane. 2020. A systematic literature review on automated log abstraction techniques. *Information and Software Technology* 122 (2020), 106276. https://doi.org/10.1016/j.infsof.2020.106276

[6] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In *2009 Ninth IEEE International Conference on Data Mining*. 149–158. https://doi.org/10.1109/ICDM.2009.60

[7] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. LogMine: Fast Pattern Recognition for Log Analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (Indianapolis, Indiana, USA) *(CIKM '16)*. Association for Computing Machinery, New York, NY, USA, 1573–1582. https://doi.org/10.1145/2983323.2983358

[8] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. 2018. Towards Automated Log Parsing for Large-Scale Log Data Analysis. *IEEE Transactions on Dependable and Secure Computing* 15, 6 (2018), 931–944. https://doi.org/10.1109/TDSC.2017.2762673

[9] Pinjia He, Jieming Zhu, Pengcheng Xu, Zibin Zheng, and Michael R. Lyu. 2018. A Directed Acyclic Graph Approach to Online Log Parsing. arXiv:1806.04356 [cs.SE]

[10] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*. 33–40. https://doi.org/10.1109/ICWS.2017.13

[11] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. 2021. A Survey on Automated Log Analysis for Reliability Engineering. *ACM Comput. Surv.* 54, 6, Article 130 (July 2021), 37 pages. https://doi.org/10.1145/3460345

[12] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2020. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics. arXiv:2008.06448 [cs.SE]

[13] Shaohan Huang, Yi Liu, Carol Fung, Rong He, Yining Zhao, Hailong Yang, and Zhongzhi Luan. 2020. Paddy: An Event Log Parsing Approach using Dynamic Dictionary. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. 1–8. https://doi.org/10.1109/NOMS47738.2020.9110435

[14] Zhen Ming Jiang, Ahmed E. Hassan, Parminder Flora, and Gilbert Hamann. 2008. Abstracting Execution Logs to Execution Events for Enterprise Applications (Short Paper). In *2008 The Eighth International Conference on Quality Software*. 181–186. https://doi.org/10.1109/QSIC.2008.50

[15] Staffs Keele et al. 2007. *Guidelines for performing systematic literature reviews in software engineering*. Technical Report. Citeseer.

[16] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2012. A Lightweight Algorithm for Message Type Extraction in System Application Logs. *IEEE Transactions on Knowledge and Data Engineering* 24, 11 (2012), 1921–1936. https://doi.org/10.1109/TKDE.2011.138

[17] Weibin Meng, Ying Liu, Federico Zaiter, Shenglin Zhang, Yihao Chen, Yuzhe Zhang, Yichen Zhu, En Wang, Ruizhi Zhang, Shimin Tao, Dian Yang, Rong Zhou, and Dan Pei. 2020. LogParse: Making Log Parsing Adaptive through Word Classification. In *2020 29th International*

Conference on Computer Communications and Networks (ICCCN). 1–9. https://doi.org/10.1109/ICCCN49398.2020.9209681

[18] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A Search-Based Approach for Accurate Identification of Log Message Formats. In *Proceedings of the 26th Conference on Program Comprehension* (Gothenburg, Sweden) *(ICPC '18)*. Association for Computing Machinery, New York, NY, USA, 167–177. https://doi.org/10.1145/3196321.3196340

[19] Haibo Mi, Huaimin Wang, Yangfan Zhou, Michael Rung-Tsong Lyu, and Hua Cai. 2013. Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems. *IEEE Transactions on Parallel and Distributed Systems* 24, 6 (2013), 1245–1255. https://doi.org/10.1109/TPDS.2013.21

[20] Masayoshi Mizutani. 2013. Incremental Mining of System Log Format. In *2013 IEEE International Conference on Services Computing*. 595–602. https://doi.org/10.1109/SCC.2013.73

[21] Meiyappan Nagappan and Mladen A. Vouk. 2010. Abstracting log lines to log event types for mining software system logs. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 114–117. https://doi.org/10.1109/MSR.2010.5463281

[22] Keiichi Shima. 2016. Length Matters: Clustering System Log Messages using Length of Words. arXiv:1611.03213 [cs.OH]

[23] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating System Events from Raw Textual Logs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management* (Glasgow, Scotland, UK) *(CIKM '11)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2063576.2063690

[24] R. Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations Management (IPOM 2003) (IEEE Cat. No.03EX764)*. 119–126. https://doi.org/10.1109/IPOM.2003.1251233

[25] Risto Vaarandi and Mauno Pihelgas. 2015. LogCluster - A data clustering and pattern mining algorithm for event logs. In *2015 11th International Conference on Network and Service Management (CNSM)*. 1–7. https://doi.org/10.1109/CNSM.2015.7367331

[26] Pengyu Wen, Zhizhong Zhang, and Bingguang Deng. 2020. OLMPT: Research on Online Log Parsing Method Based on Prefix Tree. In *Proceedings of the 3rd International Conference on Information Technologies and Electrical Engineering* (Changde City, Hunan, China) *(ICITEE2020)*. Association for Computing Machinery, New York, NY, USA, 55–59. https://doi.org/10.1145/3452940.3452951

[27] Wikipedia contributors. 2021. Logging (software) — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Logging_(software)&oldid=1042915159 [Online; accessed 22-October-2021].

[28] Tong Xiao, Zhe Quan, Zhi-Jie Wang, Kaiqi Zhao, and Xiangke Liao. 2020. LPV: A Log Parser Based on Vectorization for Offline and Online Log Parsing. In *2020 IEEE International Conference on Data Mining (ICDM)*. 1346–1351. https://doi.org/10.1109/ICDM50108.2020.00175

[29] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. 2019. Tools and Benchmarks for Automated Log Parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 121–130. https://doi.org/10.1109/ICSE-SEIP.2019.00021

A List of selected papers

In this section you can find the papers that were selected for the survey. The title of each subsection represents the query used in Google Scholar or in Scopus. Each subsection contains the papers selected for the respective query. Where applicable, the sublists for each of the papers selected represent papers considered to be relevant during backward snowballing.

A.1 Query 1: "log parsing"

This query was used in Google Scholar. From the first 100 results, 27 were selected. An additional 29 were selected during the process of backward snowballing.

1. Tools and Benchmarks for Automated Log Parsing
    1.1. SherLog: Error Diagnosis by Connecting Clues from Run-time Logs
    1.2. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning
    1.3. Detecting Large-Scale System Problems by Mining Console Logs
    1.4. A Data Clustering Algorithm for Mining Patterns From Event Logs
    1.5. LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs
    1.6. Clustering Event Logs Using Iterative Partitioning
    1.7. Length Matters: Clustering System Log Messages using Length of Words
    1.8. LogMine: Fast Pattern Recognition for Log Analytics
    1.9. Abstracting Log Lines to Log Event Types for Mining Software System Logs
    1.10. LogSig: Generating System Events from Raw Textual Logs
    1.11. Incremental Mining of System Log Format
    1.12. https://ieeexplore.ieee.org/document/4601543
2. Towards Automated Log Parsing for Large-Scale Log Data Analysis
    2.1. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis
    2.2. A Lightweight Algorithm for Message Type Extraction in System Application Logs
3. An Evaluation Study on Log Parsing and Its Use in Log Mining
    3.1. Mining Event Logs with SLCT and LogHound
4. Drain: An Online Log Parsing Approach with Fixed Depth Tree
5. A Directed Acyclic Graph Approach to Online Log Parsing
6. Logram: Efficient Log Parsing Using n-Gram Dictionaries
    6.1. Mining Invariants from Console Logs for System Problem Detection
    6.2. An automated approach for abstracting execution logs to execution events
    6.3. Efficiently Extracting Operational Profiles from Execution Logs Using Suffix Arrays
7. Self-Supervised Log Parsing
8. LogParse: Making Log Parsing Adaptive through Word Classification

8.1. Learning Latent Events from Network Message Logs
9. Improving Performances of Log Mining for Anomaly Prediction Through NLP-Based Log Parsing
10. Spell: Streaming Parsing of System Event Logs
10.1. LogTree: A Framework for Generating System Events from Raw Textual Logs
10.2. HLAer: a System for Heterogeneous Log Analysis
11. LPV: A Log Parser Based on Vectorization for Offline and Online Log Parsing
12. An Efficient Log Parsing Algorithm Based on Heuristic Rules
13. Paddy: An Event Log Parsing Approach using Dynamic Dictionary
14. A Theoretical Framework for Understanding the Relationship Between Log Parsing and Anomaly Detection
15. Spell: Online Streaming Parsing of Large Unstructured System Logs
16. A Confidence-Guided Evaluation for Log Parsers Inner Quality
17. AWSOM-LP: An Effective Log Parsing Technique Using Pattern Recognition and Frequency Analysis
17.1. Towards an NLP-based log template generation algorithm for system log analysis
18. Prefix-Graph: A Versatile Log Parsing Approach Merging Prefix Tree with Probabilistic Graph
18.1. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs
18.2. Logan: A Distributed Online Log Parser
19. Efficient and Robust Syslog Parsing for Network Devices in Datacenter Networks
19.1. Device-Agnostic Log Anomaly Classification with Partial Labels
20. Robust Log-Based Anomaly Detection on Unstable Log Data
20.1. Experience Report: Log Mining Using Natural Language Processing and Application to Anomaly Detection
21. LogStamp: Automatic Online Log Parsing Based on Sequence Labelling
22. A Review of Unstructured Data Analysis and Parsing Methods
23. OLMPT: Research on Online Log Parsing Method Based on Prefix Tree
24. A Parallel Approach of Weighted Edit Distance Calculation for Log Parsing
24.1. LogMaster: Mining Event Correlations in Logs of Large-scale Cluster Systems
25. Flexible Log File Parsing using Hidden Markov Models
25.1. A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs
26. Log Clustering Based Problem Identification for Online Service Systems
26.1. Experience Mining Google's Production Console Logs
27. Unsupervised Noise Detection in Unstructured data for Automatic Parsing

A.2 Query 2: "log parsing survey"

This query was used in Google Scholar. From the first 100 results, 2 were selected. An additional 2 were selected during the process of backward snowballing.

1. System log clustering approaches for cyber security applications: A survey
1.1. One Graph Is Worth a Thousand Logs: Uncovering Hidden Structures in Massive System Event Logs
1.2. GenLog: Accurate Log Template Discovery for Stripped X86 Binaries
2. A Survey on Automated Log Analysis for Reliability Engineering

A.3 Query 3: "log abstraction"

This query was used in Google Scholar. From the first 100 results, 7 were selected. An additional 2 were selected during the process of backward snowballing.

1. A systematic literature review on automated log abstraction techniques
1.1. A Method of Large - Scale Log Pattern Mining
2. Symptom-based Problem Determination Using Log Data Abstraction
3. Unsupervised Event Abstraction using Pattern Abstraction and Local Process Models
4. Event-Log Abstraction using Batch Session Identification and Clustering
5. Event Log Abstraction in Client-Server Applications
6. Log Abstraction for Information Security: Heuristics and Reproducibility
6.1. amulog: A General Log Analysis Framework for Diverse Template Generation Methods
7. Practical Multi-pattern Matching Approach for Fast and Scalable Log Abstraction

A.4 Query 4: "log abstraction survey"

This query was used in Google Scholar. From the first 100 results, 0 were selected.

–

A.5 Query 5: "event log parsing"

This query was used in Google Scholar. From the first 100 results, 6 were selected. One additional was selected during the process of backward snowballing.

1. LogLens: A Real-Time Log Analysis System
2. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics
3. Experience Report: System Log Analysis for Anomaly Detection
4. LOGAIDER: A Tool for Mining Potential Correlations of HPC Log Events

5. LogGAN: a Log-level Generative Adversarial Network for Anomaly Detection using Permutation Event Modeling

5.1. Event Extraction from Streaming System Logs

6. A Search-based Approach for Accurate Identification of Log Message Formats

A.6 Query 6: "log signature extraction"

This query was used in Google Scholar. From the first 100 results, 6 were selected. Zero additional papers were selected during the process of backward snowballing.

1. Unsupervised Signature Extraction from Forensic Logs
2. Towards a neural language model for signature extraction from forensic logs
3. A hybrid approach for log signature generation

A.7 Query 7: "event log signature extraction"

This query was used in Google Scholar. From the first 100 results, 0 were selected.

–

A.8 Query 1: "TITLE-ABS-KEY(log AND parsing) OR ((logs OR log OR logging OR events OR "event log" OR "event logs" OR "event logs templates" OR "event log signatures") AND (abstractionOR parsing))"

This query was used in Scopus. From all 392 results, 13 were selected. Zero additional papers were selected during the process of backward snowballing.

1. Log and Execution Trace Analytics System
2. Virtual Knowledge Graphs for Federated Log Analysis
3. The Use of Template Miners and Encryption in Log Message Compression
4. LogEA: Log Extraction and Analysis Tool to Support Forensic Investigation of Linux-based System
5. On Automatic Parsing of Log Records
6. MoniLog: An Automated Log-Based Anomaly Detection System for Cloud Computing Infrastructures
7. An Improved KNN-Based Efficient Log Anomaly Detection Method with Automatically Labeled Samples
8. An Extensible Parsing Pipeline for Unstructured Data Processing
9. A Dynamic Processing Algorithm for Variable Data in Intranet Security Monitoring
10. METING: A Robust Log Parser Based on Frequent n-Gram Mining
11. Log Parser with One-to-One Markup
12. FastLogSim: A Quick Log Pattern Parser Scheme Based on Text Similarity
13. AECID-PG: A Tree-Based Log Parser Generator To Enable Log Analysis

B Experiments visualisations

This section contains visualisations of runtime and memory consumption measurements for IPLoM, AEL, Lenma, MoLFI, SHISO, LogCluster, LogSig, and LKE.

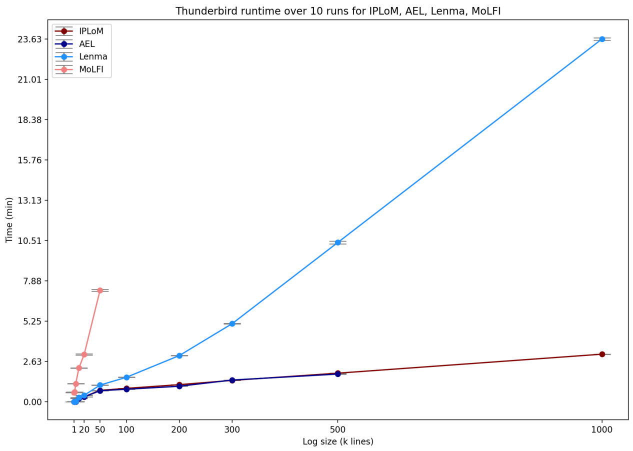


Figure 14. Runtime measurements using the Windows dataset for IPLoM, AEL, Lenma, and MoLFI.

In Figure 14 we can observe that there is a significant difference between IPLoM & AEL and Lenma. Compared to the first two that parse the dataset in about 1.5 minutes (for 500k log lines), Lenma parses 500k in 10 minutes. Also, in Figure 15 we notice a significant difference between the aforementioned methods and MoLFI. The first three methods parse 50k logs in roughly 1 minute, compared to MoLFI which takes 8.

We plot the same measurements for the Windows dataset.

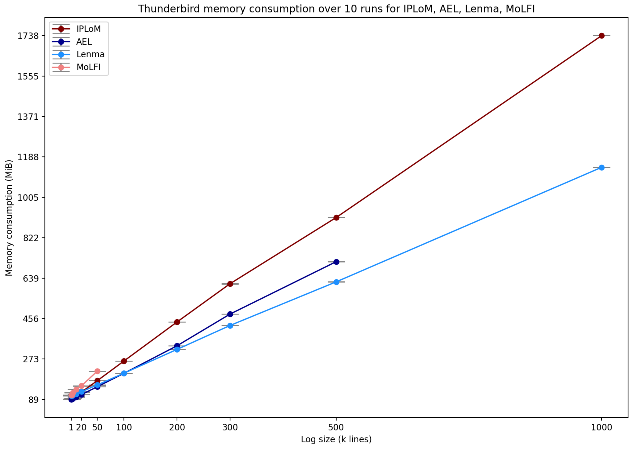


Figure 15. Memory consumption measurements using the Thunderbird dataset for IPLoM, AEL, Lenma, and MoLFI.

These can be found in Figure 16 and Figure 17 respectively.

It can be see that the methods perform similarly to the Thunderbird dataset. However, we notice some differences. For the Windows dataset, the runtime of Lenma reduces significantly compared to the previous scenario. Also. MoLFI runtime seems to increase, compared to the experiments performed on Thunderbird.

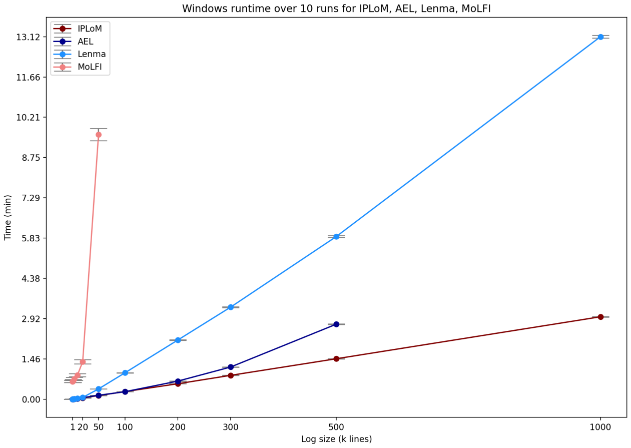


Figure 16. Runtime measurements using the Windows dataset for IPLoM, AEL, Lenma, and MoLFI.

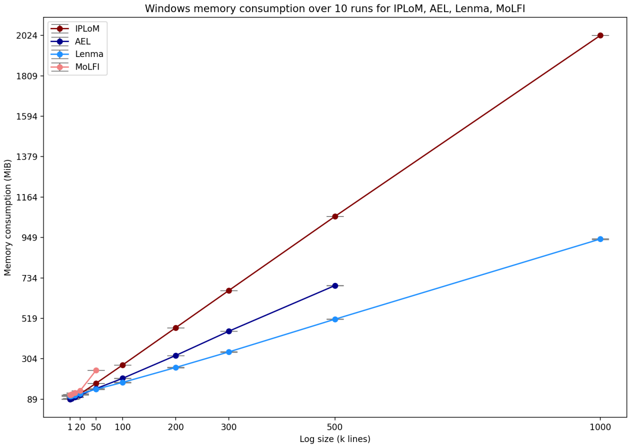


Figure 17. Memory consumption measurements using the Windows dataset for IPLoM, AEL, Lenma, and MoLFI.