Heuristic-based Parsing System for Big Data Log

Teng Li*, Shengkai Zhang†, Yebo Feng‡, Jiahua Xu§, Zhuo Ma¶, Yulong Shen‖, Jianfeng Ma**
*†¶‖**Xidian University, ‡Nanyang Technological University,§University College London
*†¶‖**State Key Laboratory of Integrated Services Networks (ISN)
Email: *litengxidian@gmail.com

GLOBECOM 2024 - 2024 IEEE Global Communications Conference | 979-8-3503-5125-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/GLOBECOM52923.2024.10901555

***Abstract*—Logs play a crucial role in recording valuable system runtime information, extensively utilized by service providers and users for effective service management. A typical approach in service management, based on log analysis, involves parsing the original log messages initially presented in an unstructured format. Subsequently, a data mining model is employed to extract critical system behavior information, aiding in service management. As the volume of logs rapidly increases, training models using current log resolution methods post-log collection becomes excessively time-consuming, leading to decreased accuracy. Manual analysis of extensive logs is both time-intensive and inefficient. This article introduces Aclog, an automated log parsing tool tailored for large-scale log analysis, storage, and management. Aclog operates by storing and managing logs in a structured and unified format, thereby offering a cohesive database for comprehensive log auditing of computing systems. Key components of Aclog encompass the log updater, log parser, log storage, and log querier. In this paper, we utilize a real-world, large-scale public log dataset to showcase the capabilities of Aclog. We evaluate the log files generated by ten popular systems.**

***Index Terms*—Big data, Log parsing, Log analysis**

I. Introduction

Logs play crucial roles in computer systems by providing a detailed record of operational activities. They enable debugging, troubleshooting, and analysis of security or performance issues. Despite their valuable content, logs pose significant challenges for analysis and processing due to their vast amounts of unstructured and complex semantics and low-level message types.

Previous research has demonstrated that parsing structured log templates can effectively assist in log analysis. These approaches can parse logs to reduce storage requirements, structure data to facilitate system operation rule analysis, and parse parameters into tables to streamline log management. Numerous high-precision methods have been proposed for parsing structured logs. For example, Du et al. proposed Spell [1], which leverages the longest common sub-sequence to parse, manage, and analyze the system event logs; He et al. proposed Drain [2], a method that employs a fixed depth parse tree to parse logs continuously; Palkar et al. [3] optimize the log analysis workflow by applying filters to the raw byte stream of logs before parsing; Bonifati et al. [4] parse structured knowledge by querying logs in large databases.

However, balancing log parsing accuracy with the storage space and analysis time consumption presents a significant challenge when handling a large volume of logs. Therefore, there is a pressing need for a new method that efficiently and accurately parses unstructured logs, without sacrificing log granularity information.

Building an automatic tool to solve the aforementioned problems and help log analysis presents three major challenges:

1) ***Large Throughput of Log Data:*** An enterprise-level system can easily generate gigabytes of log data per hour, resulting in a substantial accumulation of data over time. Therefore, efficiently parsing log templates is necessary for a usable log analysis system.
2) ***Complex and Unstructured Log Semantics:*** Logs are usually recorded using free text, resulting in a complex and unstructured semantic structure with low-level message types. The resolution accuracy of algorithms used to parse information from different log types varies, and there exists no universal algorithm for this problem.
3) ***Changes in Log Templates:*** While log templates can be obtained from developers' support pages or manuals, changes resulting from software updates can render them outdated. Additionally, most companies do not disclose the source code of log systems, making it challenging to parse log templates without prior knowledge.

In this paper, we propose and implement an automatic log template parsing approach, Aclog. It can efficiently parse unstructured log messages into structured log templates and dynamic parameters to help further analysis and system management.

Aclog adopts a hierarchical log tree approach to classify logs, serving as the foundation for its analysis. By concurrently examining three levels of tree nodes, Aclog identifies dynamic variables and employs an optimization technique to accurately extract template information. With a time complexity close to $O(n)$, Aclog operates with exceptional efficiency, surpassing existing log template parsing methods. The generated log templates and parameter types offer a succinct and intuitive overview of extensive logs, providing structured data that is valuable for downstream data analysts.

II. Methodology

In this section, we will detail the data structure of our method and point out its workflow.

Corresponding author: Teng Li.

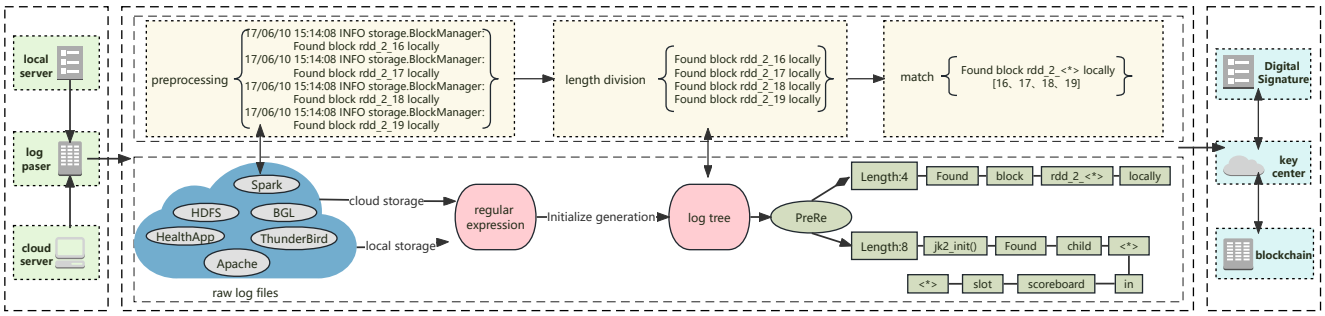


Fig. 1: Overall architecture of the proposed method

A. Architecture

Aclog stands as an automated log parsing tool comprising three integral components: the log updater, log parser, and log querier, alongside log storage. The log collector, the initial component, imports local logs from external computer systems into the Aclog framework. Central to Aclog is the log parser, pivotal for subsequent audits. It meticulously identifies each original log statement, juxtaposing it against the template within the log tree, thereby dissecting it into log templates and parameters. During runtime, the log updater dynamically modifies corresponding templates and parameters, optimizing efficiency and accuracy. Log parameters find their abode within respective log templates in the database. Subsequently, the log querier loads logs from the database into the application, facilitating user queries or visual representations.

In the ensuing section, we will delineate Aclog's methodology in parsing original log messages, followed by an elucidation of its underlying data structure—the log-tree.

B. Data Structure

The basic data structure of Aclog comprises PreRe, which includes simple regular expression filtering at the top layer. Filtered log messages are then divided into different branches based on their length, with each length forming a log-tree. The vertex in the log tree represents the length of the log messages. Upon observing the original logs, it was found that most logs generated by the same print template had only different parameter positions. Therefore, in PreRe, logs are preliminarily classified by calculating their size, assuming that logs with the same template have the same length. Other nodes in the log tree are internal nodes with special coding rules. At the end of each path in the log tree, information about all nodes in the path is saved, including templates and a set of dynamic variables. Template information is viewed without backtracking after matching the best template.

For a node inside the log tree with a length of m, let s represent a set of strings, such as $10-9$ or $a-z$. After passing through the regular expression, the original log message is decomposed into a set of rows represented by such strings. Let T be a set of m strings, i.e., $T = s_1, s_2, ..., s_m$. In the internal nodes of the log tree, $s_1, s_2, ..., s_m$ are connected end to end in the form of tree nodes. Any node s_n $(0 < n < m + 1)$ may have a node branch. When matching the current node s_n as a dynamic variable, the node becomes $*$ to represent the parameters. Nodes in the same layer are stored in the form of dictionaries, making it convenient for searching and comparison. Given any pre-resolved log message $a = ax_1, ax_2, ax_3, ..., am$ $(x_i \epsilon \sum, 1 < i < m)$, for example, "ashe", the log tree is searched for a Log template match to find the position of all strings in any string T given a specific set of strings.

C. Workflow

As illustrated in Figure 1, inspired by the process of manual log classification, Aclog uses a heuristic-based tree structure to parse logs in real-time. Initially, the log tree is empty. When an original log message arrives, it is first divided into a log string collection using predefined delimiters, and its length is classified into log trees of corresponding length. Following this, Aclog proceeds to scour all nodes of the log tree in search of the current sequence string. Should no match be uncovered, the system undertakes an evaluation to ascertain if a new template necessitates creation and subsequent insertion into the log tree. In the subsequent segment of this subsection, we will meticulously expound upon each step.

Data Cleansing and Preparation. Preprocessing is a crucial step in log parsing, which can improve the efficiency and accuracy of subsequent steps. However, due to the complexity and variability of log attributes, too many pre-definitions can weaken the portability of unsupervised parsing. Therefore, our efforts are focused on reducing the impact of data preprocessing on the downstream data processing steps.

By extracting information pieces by spaces and storing them with lists, Aclog can decompose consecutive log statements into a sequence of words without utilizing regular expressions. However, to make Aclog directly comparable to other log parsers, we also add regular expression support to Aclog. In the regular-expression mode, after the original log message arrives, Aclog leverages publicly defined regular expressions in the LogPai benchmark to identify log keywords or parameters. For example, Some common variables in the original log statement, such as IP addresses and blocks, can be easily identified as parameters using "$blk[0-9]+$".

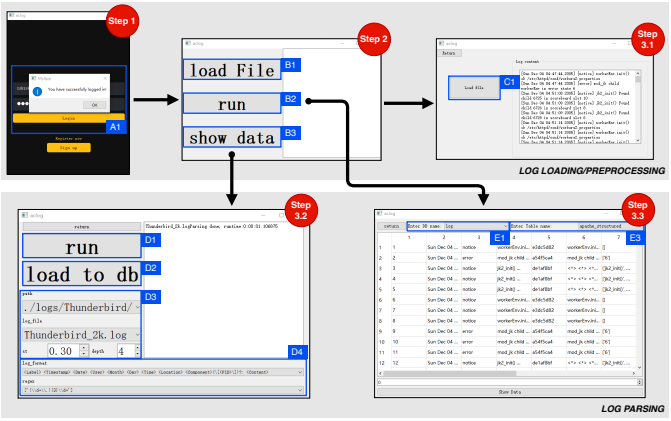


Fig. 2: User interface of Aclog

TABLE I: Data Set for Evaluation.

Dataset	Description	Size	Messages	Templates
HDFS	Hadoop distributed file system log	1.47GB	11,175,629	14
Hadoop	Hadoop MapReduce job log	48.61 MB	394,308	114
Spark	Spark job log	2.75 GB	33,236,604	36
ZooKeeper	ZooKeeper service log	9.95 MB	74,380	50
Linux	Linux system log	2.25 MB	25,567	27
Mac	Mac OS log	16.09 MB	117,283	341
Apache	Apache server error log	4.90 MB	56,481	6
BGL	Blue Gene/L supercomputer log	708.76 MB	4,747,963	589
HealthApp)	Health app log	22.44 MB	253,395	221
Thunderbird	supercomputer log	29.60 GB	211,212,192	899

String Pattern Matching. After the log preprocessing step, the original log message is expressed as a sequence of words with a length of $Length$. Aclog starts from the PreRe root node and classifies the log message using the string collection length, based on the assumption that log messages with the same log template have the same length. For instance, "The current CPU temperature is 45°" has a string set length of 7 and will be divided into a log tree of $Length$: 7. Although log messages with the same log event may have different log message lengths, they can be handled easily. In Aclog, string sets that are not matched successfully are matched again from $Length$: 6 and $Length$: 8. Whether to match from different lengths is controlled by specific experimental parameters. If it remains unsuccessful, Aclog will create a new pattern string.

Aclog's automatic matching mechanism determines which log tree the current log string set needs to find the template in. Specifically, nodes in the same layer are stored in dictionaries. Aclog takes three strings from the log string set in order and matches them with nodes in the three layers. There are two cases when comparing a string with a child node: either the corresponding node is the same, or it is different. Traditional algorithms will consider a mismatch and move to the next node instead of taking into account the presence of parameters in the log. Aclog matches three nodes at a time. If they are the same, the same sub-node is defined as the parameter $< * >$, and three more nodes in the set are compared. This step is repeated until all matching is completed and it is determined that they have the same log template. If all rounds are different, it is considered that this is not the same template.

Automated Log Template Enhancement. After matching the appropriate log template in pattern string matching, Aclog will add the dynamic parameters of the current log message to the LogVariable in the log template. Additionally, the log events in the returned log template will be updated. Specifically, Aclog will determine whether any node changes to the parameter $*$ node on the path of the current tail node and follow the corresponding template location in the new tail node. When the new dynamic parameter $< * >$ is the first or last of the log message, Aclog's parsing accuracy will not be affected. Conversely, tree structures that calculate similarity will not be judged in this case.

When the corresponding log template is not matched in pattern string matching, Aclog creates a new template based on the current log message. Its dynamic parameters are empty, and there is no $*$ on the log node path since the existing template has only itself. Aclog initiates the creation of fresh log tree branches utilizing the derived log template. This process involves traversing from the root node to the designated node that warrants inclusion of the new log template, while simultaneously incorporating any absent child nodes along the traversal path. Ultimately, Aclog navigates a path along the template path, identifying the subsequent matching child node for each node, and proceeds to update the dictionary accordingly.

D. User Interface

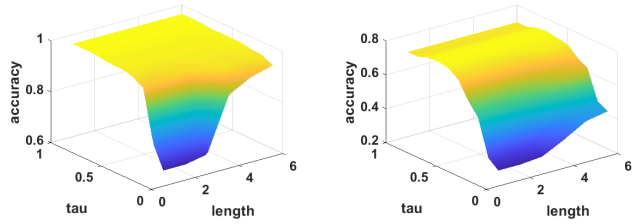
We provide an interactive application interface as the front end of Aclog. Figure 2 illustrates the user interface of Aclog. In this demonstration, we use the LogHub [5] benchmark. Aclog can be applied to other database systems with simple extensions. Aclog's design integrates login, registration, parsing, upload, and query components to address challenges. Authentication leads users to the main interface for local file querying, parsing, storage, and database log retrieval. Aclog operates in three steps: loading local logs, parsing with user-defined parameters, and querying structured logs from the database. Aclog automates real-time collection, parsing, and storage of system logs, accessible only to authorized administrators for auditing and analysis. This automated process ensures accurate large-scale log handling without loss of granularity.

III. Evaluation

In this section, we evaluate our approach from multiple aspects, including the parameter setup, classification efficacy, deployability, and operation efficiency.

A. Dataset

We employed log files sourced from ten popular systems, selected from the benchmark set provided by Logpai [6]. Comprehensive details are presented in Table I. These logs, originating from real servers, encompass a wide array of



(a) BGL (b) MAC

Fig. 3: Parameter threshold Settings

log types, including distributed system logs such as HDFS [7], Hadoop [8], Spark [9], and ZooKeeper [10], alongside operating system logs like Linux [11] and Mac [12]. Furthermore, they include server application logs like Apache [13], supercomputer logs including BGL [14] and ThunderBird [15], and mobile phone system logs such as HealthApp [16]. These datasets are derived from open-source logs spanning various domains, as well as logs generated by industrial systems. They have seen extensive use in previous log analysis and research endeavors, including tasks such as system exception detection, problem diagnosis, and system comprehension.

Each dataset within the Logpai collection comprises a subset of 2000 manually categorized log messages, meticulously organized into log events to facilitate the assessment of our log parser's accuracy. To evaluate the efficacy of log resolution, we utilize the complete log of each dataset. To illustrate the effectiveness of Aclog, we compare its performance with the existing six log parsing methods(Drain [2], Spell [1], IPLoM [17], AEL [18], LFA [19], Lenma [20]) in terms of accuracy, efficiency, and effectiveness.

B. Setup

When Aclog parses logs, it sets two parameters to control the accuracy and time of log parsing: length and threshold. After the preset matching length or the current matching accuracy is exceeded, the Aclog will end this round of matching, indicating that the corresponding template of the log has been found. Different lengths and thresholds will have a significant impact on the resolution results. The shorter the length and the lower the threshold, the faster the matching speed. Due to the diversity of each system, Aclog will have different performances when facing other systems. That is, Aclog has different analytic lengths and thresholds for different logs. To find the best length and threshold, we have analyzed the resolution of different lengths and thresholds for ten logs. As shown in Figure 3, we select two MAC logs and BGL logs whose resolution accuracy is greatly affected by the length and threshold for display. The lighter the color, the higher the accuracy. As you can see, the accuracy increases with the length and threshold value increase. Considering its impact on efficiency, we select the point with the minimum length plus threshold synthesis as a reference for subsequent efficiency comparison.

C. Detection Efficacy

F-measure is a statistic, F-Measure is also called F-Score, F-Measure is Precision and Recall weighted harmonic average and is a commonly used evaluation standard in the IR (information retrieval) field, which is often used to evaluate the quality of classification models. We use F-measure, a commonly used evaluation standard in the information field, to evaluate the quality of classification models. In the f-measure function, when the parameter is 1, F1 combines precision and recall results. When F1 is high, the test method is more effective. As shown in Table II, in the Aclog experiment, the f-measure is above 0.98, which shows that our test is effective enough.

It can be seen from the Table III that the accuracy of the log parsing method proposed in this paper, namely Aclog, is better than that of the seven methods in ten logs. The special rules of offline IPLoM make it obtain high precision on most log data sets. Both online methods, Spell and Drain, have high-resolution accuracy, and their similar prefix tree structures have achieved high precision on HDFS and Apache. LenMA uses the word length as the analysis vector, which has the worst accuracy on the Health App. LFA is not suitable for Linux. Aclog is the best in general because it simulates the way of manually classifying logs and uses three special rules: first, logs with the same template have the same length, which makes use of the log length to classify the original logs efficiently. Secondly, the log parameters with the same template have the same adjacent locations, which provides a basis for matching layer three nodes at the same time. Third, logs with the same prefix more than a certain proportion have the same template. After introducing the same prefix proportion, you can quickly cluster logs without comparing the entire log template.

D. runtime evaluation

Because Drain sets the maximum child node, it will not cause a profound parse tree during parsing, which greatly reduces the matching time. However, this may affect the accuracy of parsing in some cases. Spell also reduces the running time by using the prefix tree to store log events, but Spell uses the longest common subsequence method for comparison, which is time-consuming. During log parsing, Aclog will create a template for each type of log, which will become an important factor affecting the running time when the log data volume is small. As the number of logs increases, the log template is gradually learned by Aclog. When the number of log templates is stable, the running time of Aclog for a single log is O(n), and n is the log length. Therefore, the parsing time of Aclog is not limited to large-scale logs. The running time of Aclog still increases linearly when the number of daily logs increases sharply. Aclog is not limited by memory because it can parse logs.

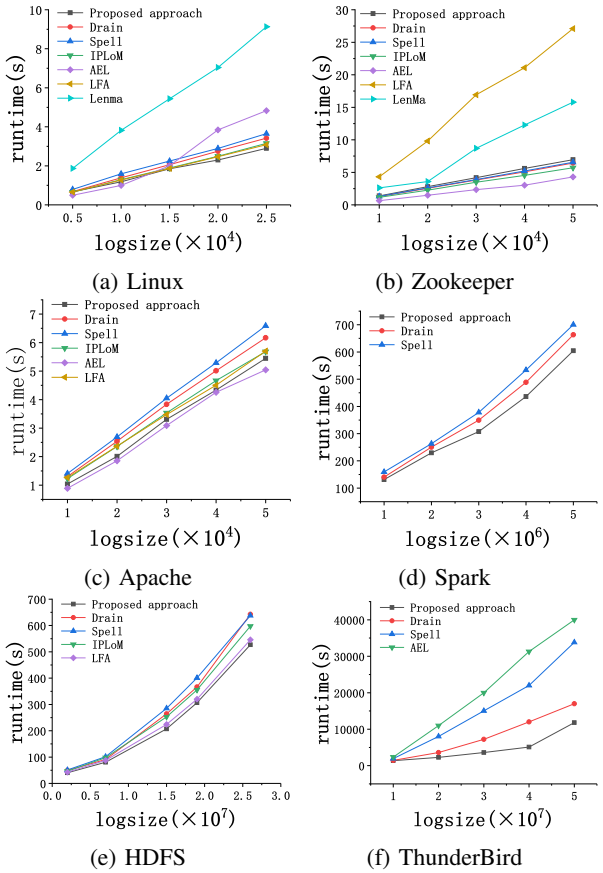
Through experimental comparison in Figure 4, we can see that Aclog has no advantage over other online log parsers when the data volume is small. Within 1w logs, parsing can be completed within 10s. When the log volume becomes larger,

TABLE II: F1 measure, Precision, and Recall for Evaluation

	HDFS	HealthApp	Thunderbird	Zookeeper	BGL	Hadoop	Apache	Mac	Linux	Spark
F1 measure	99.99%	98.03%	99.98%	99.97%	99.97%	99.63%	1	80.64%	99.24%	99.17%
Precision	1	98.76%	99.98%	99.96%	99.95%	99.96%	1	85.74%	90.73%	98.35%
Recall	1	99.71%	99.98%	99.98%	1	1	1	99.1%	99.48%	1

TABLE III: Accuracy Performance comparison between the Aclog and other algorithms.

	HDFS	HealthApp	Thunderbird	Zookeeper	BGL	Hadoop	Apache	Mac	Linux	Spark
Spell	1.000	0.639	0.844	0.964	0.787	0.778	1	0.757	0.605	0.905
Drain	0.998	0.780	0.955	0.967	0.963	0.948	1	0.787	0.690	0.920
AEL	0.998	0.568	0.941	0.921	0.957	0.869	1	0.764	0.673	0.905
LFA	0.885	0.549	0.649	0.839	0.854	0.900	1	0.599	0.279	0.994
IPLoM	1.000	0.825	0.663	0.962	0.939	0.954	1	0.671	0.672	0.920
LenMa	0.998	0.174	0.943	0.841	0.690	0.885	1	0.698	0.701	0.884
Aclog	0.998	0.895	0.95	0.982	0.965	0.972	1	0.72	0.733	0.98



(a) Linux (b) Zookeeper (c) Apache (d) Spark (e) HDFS (f) ThunderBird

Fig. 4: Runtime comparison

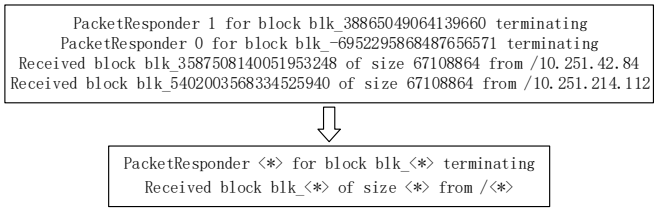


Fig. 5: Example of a log parsing template

the advantage of Aclog is obvious. It can be seen that with a log volume of 10^5, Aclog is close to the fastest-running LFA. Even the fastest Drain has thousands of seconds more running time than Aclog under the data volume of 10^6. Aclog greatly improves the running time of existing online parsing methods on large-scale log data. On the evaluation of six log data, Aclog shows excellent performance, and the running time on HDFS is increased by 0.08 (Aclog and Drain are on HDFS $3*10^5$); On Thunderbird, the running time of Aclog and Drain is increased by 0.637 (on Thunderbird $3 * 10^7$).

IV. RELATED WORK

Mining log templates from logs has been a vibrant area of research. SLCT [21] stands out as one of the earliest log clustering algorithms, grouping log lines sharing the same pattern into clusters. Makanju et al. [17] introduced iterative methods for partitioning system logs in IPLoM without relying on prior knowledge. SHISO is a novel approach for mining log formats and parameters in incremental log messages, constructing a structured tree from log message-derived nodes. Vaarandi et al. [22] presented LogCluster, which identifies frequent line patterns and outliers from text event logs. Hamoni et al. [23] introduced LogMine, a hierarchical log parser within the map simplification framework of distributed platforms. Spell utilizing the longest common subsequence-based method for parsing system event logs and managing discovered message types in stream mode. HE et al. proposed Drain, employing a fixed depth parse tree for streaming and timely log parsing. Messaoudi et al. [24] introduced the MoLFI method, redefining log message recognition as a multi-objective problem and employing the NSGA II algorithm to search the solution space of Pareto optimal message template sets.

Comparison reveals that IPLoM [17], SHISO [25], LogCluster [22], LogMine [23], etc., primarily focus on offline structured batch processing or matching new log templates with message types and regular expressions offline. Spell [1] parses log messages using the longest common subsequence algorithm, contributing significantly to online parsing. However, dependency on LCS results in increased matching time

with a higher volume of logs. Drain [2] employs a fixed depth tree to represent hierarchical relationships between log messages, defining grouping rules for each layer based on factors such as log message length and token similarity. Zhu et al. [6] evaluated the performance of various data mining approaches, finding Drain to excel in terms of accuracy and efficiency. Logram [26] decomposes log messages into n-gram word vectors stored in a dictionary, distinguishing constant templates from dynamic variables. While Logram achieves O(1) template comparison, it consumes significant storage space.

V. Conclusion

In summary, this paper introduces Aclog, an advanced log parsing method that excels in both precision and security. It outperforms existing benchmarks like IPLoM and LFA in accuracy and clustering, making it a valuable tool for system monitoring, cybersecurity, and network analysis. Future enhancements should focus on adapting to dynamic log environments, scaling for large datasets, and improving real-time parsing capabilities. This study significantly advances log parsing methodologies, addressing contemporary priorities in data privacy and operational efficiency. Future research should refine Aclog's efficiency, precision, and applicability to evolving log structures and cybersecurity challenges.

Acknowledgments

This research is funded by the National Key Research and Development Program of China (2022YFB3103500), National Natural Science Foundation of China under Grant (No. 62272370, No. U21A20464), Young Elite Scientists Sponsorship Program by CAST (2022QNRC001), the China 111Project (No.B16037), Qinchuangyuan Scientist + Engineer Team Program of Shaanxi (No. 2024QCY-KXJ-149), the Fundamental Research Funds for the Central Universities (QTZX23071).

References

[1] M. Du and F. Li, "Spell: Online streaming parsing of large unstructured system logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2213–2227, 2018.

[2] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.

[3] S. Palkar, F. Abuzaid, P. Bailis, and M. Zaharia, "Filter before you parse: Faster analytics on raw data with sparser," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1576–1589, 2018.

[4] A. Bonifati, W. Martens, and T. Timm, "An analytical study of large sparql query logs," *The VLDB Journal*, vol. 29, no. 2-3, pp. 655–679, 2020.

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

[6] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 121–130.

[7] X. Pan, Z. Luo, and L. Zhou, "Navigating the landscape of distributed file systems: Architectures, implementations, and considerations," *arXiv preprint arXiv:2403.15701*, 2024.

[8] S. R. Zeebaree, H. M. Shukur, L. M. Haji, R. R. Zebari, K. Jacksi, and S. M. Abas, "Characteristics and analysis of hadoop distributed systems," *Technology Reports of Kansai University*, vol. 62, no. 4, pp. 1555–1564, 2020.

[9] C. Li, L. Jiang, and X. Chen, "A distributed cbir system based on dcnn on apache spark and alluxio," in *2018 International Conference on Image and Video Processing, and Artificial Intelligence*, vol. 10836. SPIE, 2018, pp. 100–105.

[10] M. Copik, A. Calotoiu, K. Taranov, and T. Hoefler, "Faaskeeper: Learning from building serverless services with zookeeper as an example," *arXiv preprint arXiv:2203.14859*, 2023.

[11] L. Chen, A. Xu, X. Kuang, H. Lv, H. Yang, Y. Yang, and B. Li, "Detecting advanced attacks based on linux logs," in *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE, 2020, pp. 60–64.

[12] A. Adekotujo, A. Odumabo, A. Adedokun, and O. Aiyeniko, "A comparative study of operating systems: Case of windows, unix, linux, mac, android and ios," *International Journal of Computer Applications*, vol. 176, no. 39, pp. 16–23, 2020.

[13] M. B. Seyyar, F. Ö. Çatak, and E. Gül, "Detection of attack-targeted scans from the apache http server access logs," *Applied computing and informatics*, vol. 14, no. 1, pp. 28–36, 2018.

[14] D. Gregor and A. Lumsdaine, "The parallel bgl: A generic library for distributed graph computations," *Parallel object-oriented scientific computing (POOSC)*, vol. 2, no. 1, 2005.

[15] A. Salamzadeh, L.-P. Dana, P. Ebrahimi, M. Hadizadeh, and S. Mortazavi, "Technological barriers to creating regional resilience in digital platform-based firms: Compound of performance sensitivity analysis and birch algorithm," *Thunderbird International Business Review*, vol. 66, no. 2, pp. 135–149, 2024.

[16] A. P. Darko, C. O. Antwi, K. Adjei, B. Zhang, and J. Ren, "Predicting determinants influencing user satisfaction with mental health app: An explainable machine learning approach based on unstructured data," *Expert Systems with Applications*, p. 123647, 2024.

[17] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 1255–1264.

[18] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 4, pp. 249–267, 2008.

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

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

[21] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*. Ieee, 2003, pp. 119–126.

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

[23] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016, pp. 1573–1582.

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

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

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