COPY RIGHT

2023 IJIEMR. Personal use of this material is permitted. Permission from IJIEMR must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. No Reprint should be done to this paper, all copy right is authenticated to Paper Authors

IJIEMR Transactions, online available on 11th Jan 2023. Link :http://www.ijiemr.org/downloads.php?vol=Volume-12&issue=Issue 01

DOI: 10.48047/IJIEMR/V12/ISSUE 01/67

Title **Yet Another Log Parser - Effective System for Pattern Recognition and Parsing of Large Log Files**

Volume 12, ISSUE 01, Pages: 703-716

Paper Authors
Srikari Rallabandi, Akash Singh

International Journal for Innovative Engineering and Management Research

USE THIS BARCODE TO ACCESS YOUR ONLINE PAPER

To Secure Your Paper As Per UGC Guidelines We Are Providing A Electronic Bar Code

Yet Another Log Parser - Effective System for Pattern Recognition and Parsing of Large Log Files

Srikari Rallabandi[1]*and Akash Singh[2]†
[1] Vidya Jyothi Institute of Technology, Hyderabad, India
sreeku.ralla@gmail.com
[2] InvenioLSI Hyderabad, India
akkiind4@gmail.com

Abstract
Software logs are the crucial cogs in software systems analysis, as such, Log files are responsible for providing useful information about software running in an organization. It is seldom suitable to store logs in databases owing to various reasons. Each of these software uses its own set of logging patterns and thus creates a different set of columnar data. These logs are often unstructured data items due to frequent changes in log lines at various levels. However, Analysis of these log messages is crucial in order to present usable information (e.g., patterns) to system administrators so that managing and monitoring the jobs in an organization is more effective, such as anomaly detection, alert generation, and event prediction. Recognizing patterns in the humongous log files from disparate sources without prior information is a huge task.

The objective of this paper is to propose a design architecture that would help recognize log patterns from heterogenous log files, thereby providing effective system from DevOPs and maintenance standpoint. In this paper, we propose a method named YALP - Yet Another Log Parser, that generates and categorizes logs into high-quality pattern efficiently with high precision rate. YALP uses map-reduce framework which can be to process millions of log messages in seconds in distributed framework. String-processing and text- processing method is applied to organize log events in groups. To identify static and dynamic content of log events, in same group, Frequency analysis is then applied. This method exploits algorithmic techniques to minimize the computational overhead since log messages are always automatically generated. All this process is done dynamically so that no log preprocessing is required from an end-user perspective. The parser is ready and outputs the structured logs as soon as it starts receiving log lines or a file containing log lines as input. Performance evaluation is then done using datasets of LogPai [16] benchmark.

Introduction

Logging in software systems is one of the most commonly used mechanisms which helps in maintenance and troubleshooting by recording the software's states and behaviours. The field of IoT(IoT involves machine-to-machine communications (M2M)) still has yet to find a standardised logging features, due to the rapid growth of the field and presence of diverse variety of "things", but continunous monitoring of interconnected devices is important so that downtime of the system is minimised by quickly detecting and resolving anomalies and bugs. Thus, it is incredibly challenging to parse and analyze log messages from software systems.

In this paper, the focus is on (semi-)automatically converting unstructured raw log events into a structured format that would facilitate further analysis. The parser will decide and categorize the log tokens into static and dynamic tokens. Log File headers are taken care of in this stage itself, as they have the same structure as other log lines. The log analyzer has one component to recognize patterns from these processed log messages and another component to match these patterns with the inflow of log messages to identify events and anomalies.

Log Analysis tools are offered by many companies, such as Splunk [23], Sumo Logic [5], Loggly [4], LogEntries [3], etc. Opensource developers have also worked in the area of log management and analysis such as ElasticSearch [1], Graylog [2], and OSSIM [10]. Regular Expressions(RegEx) is being used by the majority of these tools and packages to match log messages. These tools support supervised matching by definition, as the system administrators must have prior knowledge/experience of working with regex expressions.

Furthermore, due to the use of multiple logging libraries, new log formats are produced everyday, consequently, new regular expressions need to be added/updated on regular basis.It is impractical to use regular expressions to parse various logs, which lead researchers to develop increasingly intelligent log parsing techniques (see [12] for a good survey). Existing approaches, however, have certain drawbacks to them [12] including their reliance on domain knowledge, inability to demarcate static content from dynamic variables for complex log files, and use of advanced machine learning algorithms, which require parameter tuning at various levels.

HLAer [24] is one such methodology developed in the area of automated log pattern recognition, which parses heterogenous log messages automatically.. Owing to high memory requirements and huge overhead communication in parallel implementation, HLAer is not efficient and scalable, despite being unsupervised and robust to heterogeneity.

We propose, through this paper, YALP, an end to end framework which is capable of addressing all the deficiencies with existing tools and packages. YALP is an unsupervised framework that scans log messages only once and can quickly process hundreds of millions of log messages with minimal memory. It is a simple yet powerful approach that recognizes log structures from any log files without prior domain knowledge or using complex machine learning techniques. It relies on string matching and regional frequency analysis for data pre-processing. It begins by grouping similar log events into groups using the clustering module. A pattern is then generated for each cluster by our pattern recognition module. It then uses frequency analysis on instances of each group to distinguish between static and dynamic log message tokens. YALP works in an iterative manner that generates a hierarchy of patterns (regexes), one level at every iteration. The hierarchy allows users to select the right patterns that satisfy their needs. We implement a map-reduced version of YALP to deploy in a massively parallel data processing system.

We have organised the paper in the following different subsections: Section 2 talks about related work and background with respect to YALP. Section 3 described the proposed approach that is followed. Section 4 discusses the experimentation results and analysis. In the following Section 5, we conclude the paper, providing future scope, and work that can be done on top of YALP for complete end-to-end logging feature.

Literature Review and background

Due to the need of understanding and maintain systems during runtime, it has become increasingly important to analyze logs with better precision. As such, a lot of researchers and practitioners have started working in this area. [25] talks about the use of archived log messages to identify and troubleshoot components and system failures in High Performance Computing (HPC) and other large-scale systems. El-Masri et al. [12] shows a survey of log parsing techniques where they define a model of a classification technique and examined 17 log parsing tools existing in the area. [33] proposes a method of clustering weblogs without providing any user-defined parameters. However, because the time complexity of their algorithm is (n3), makes their method not scalable and unsuitable for large datasets. Authors of [9] propose a system where users' search logs are used to create a website's search index. [26] discusses an algorithm which does pre-processing on weblogs and extracts a set of fields from it. Similar to [9], Authors of [13] proposes a method that goes through user navigation and extracts useful information for website admins.

Vaarandi et al. [30] [31] proposed SLCT (Simple Logfile Clustering Tool), where log templates are identified using clustering techniques. SLCT analyzes frequently used terms in logs and groups them together. Dynamic tokens are analyzed and identified using density-based clustering algorithm. [32] can be considered SCLT version 2.0 written by same authors, where all the frequent terms are extracted and arranged into tuples. Then, clusters are created based on the log lines containing minimum number of tuples.

Makanju et al. [22] proposed IPLOM (Iterative Partitioning Log Mining), a log clustering approach, which uses heuristic-based hierarchical clustering. First, Log messages are partitioned based on the size of log events, which is then further subdivided based on the highest number of similar terms. Fu et al. [14] proposed LKE (Log Key Extraction), which uses a distance-based clustering technique in which weighted edit distance method are used to group log together such that if the terms appear at the beginning of events, it is given more weight. Then, LKE does the clustering of logs by providing log key to each raw log level, which is then extracted to get the common parts out and is used to generate event types.

Hamooni et al. proposed LogMine [15], where log messages generated are subjected to MapReduce algorithm to which uses MapReduce to abstract heterogeneous log messages generated from various systems. The LogMine algorithm consists of a hierarchical clustering module combined with pattern recognition. It uses regular expressions based on domain knowledge to detect dynamic variables. Then, it replaces the real value of each field with its name. It then clusters similar log messages with the friends-of-friends clustering algorithm.

Dai et al. [8] proposed Logram, which uses Natural Language Processing (NLP) Techniques, specifically n-gram dictionary to combat the need of log analysis tools which can parse large
log sizes with low parsing time. N-gram dictionary is used to differentiate between static and dynamic tokens by storing the frequencies of n-gram in logs. The biggest advantage of Logram is that it can be easily deployed in multiple core to achieve parallel processing and achieve scalability, since multiple n-gram dictionaries can be used concurrently. Template extraction from log lines can be considered as labeling sequential data problem, which is what Kobayashi et al. proposes in NLP-LTG (Natural Language Processing–Log Template Generation) [19]. Log messages are classified into static and dynamic terms using Conditional Random Fields (CRF) [20], and then build the ground truth data using regular expressions. NLM-FSE(Neural language Model-For Signature Extraction), proposed by Thaler et al. [29], classifies static and dynamic tokens by training a character-based neural network.

He et al. [16] proposed Drain, which utilizes parse tree to generate event types from log messages. First, to pick out dynamic tokens, raw log messages are pre-processed using regular expressions. Then, using the number of tokens, a parse tree is built. A similarity metric is then used which compares leaf nodes to event types, assuming that the tokens which are at the start of each log line are most likely to be static tokens.

Spell (Streaming Parser for Event Logs using an LCS) [11] proposes the idea that if same logging statement is used to produce log messages, then the Longest common subsequence of a pair of logs is a static field, and can be used as a group defining that particular cluster of logs.

Map reduce has become the go-to framework for log processing instead of storing it in DBMS, as shown in [7], where join processing techniques are demonstrated extensively using map-reduce. [21] describes how heterogeneous application logs can also exploit map-reduce framework for effective logging, and have described an architecture for it as well.

Approach followed

YALP consists of the following steps: pre-processing, grouping of similar log events, and the generation of log templates using static and dynamic tokens, which get refined as we generate a hierarchy of patterns by iteration. In the first pre-processing step, header information such as log level, timestamp and IP address is identified. We also detect trivial dynamic tokens such as IP and MAC addresses based on common regular expressions. In the second step of YALP, we identify log events that are similar and group them together using the distance function referencing LogMine [15]. This method creates a tree-like hierarchy of patterns, with general patterns being the root, and specific patterns being the child/node. Such hierarchy is useful for the system administrators to pick the right level of detail they want to track in the log messages as opposed to writing regular expressions manually. We discuss each step of breaking the log in general and specific patterns in the following subsections.

Pre-processing and Tokenization

In the first step of log parsing, log lines undergo pre-processing, where header information such as timestamp, log level and process ID which are commonly present in the first line of the log, is separated from the log. This information is fairly easy to identify using regular technical phrases [8]. In the next step of pre-processing, dynamic variables such as IP and MAC addresses are identified. As shown by He et al. [16] and all the tools studied in this paper (i.e., Drain [17], SPELL [11], Logram [8] and AEL [18]), This step is known to increase the accuracy of log parser. The grouping of similar events is discussed in more detail in the next subsection. We referred ULP [27] that created regular expressions to detect the following trivial dynamic variables: Mac addresses, IPV6 addresses, URLs (beginning with HTTP and HTTPS), numerical in hexadecimal format, Dates such as 2002-03-24 and 2002-03-24, Time in the format hh:mm:ss. All the regular expressions used in this step are generic in nature, which can cater to any log file as input.

```
1 2022-09-17 10:20:26,788 | WARN |
  Transport Connection to:
  tcp://172.25.170.139:53160 failed:
  java.io.EOFException |
  org.apache.activemq.broker.TransportCon
  nection.Transport |
ActiveMQ Transport:
  tcp:///172.25.170.139:53160@7862
2 2022-09-17 10:20:26,788 | WARN |
  Transport Connection to:
  tcp://172.25.170.139:53660 failed:
  java.io.EOFException |
  org.apache.activemq.broker.TransportCon
  nection.Transport |
ActiveMQ Transport:
  tcp:///172.25.170.139:53660@7862
3 2022-09-17 10:20:26,788 | WARN |
  Transport Connection to:
  tcp://172.25.170.139:53348 failed:
  java.io.EOFException |
  org.apache.activemq.broker.TransportCon
  nection.Transport |
ActiveMQ Transport:
  tcp:///172.25.170.139:53348@7862
4 2022-09-17 10:20:26,788 | WARN |
  Transport Connection to:
  tcp://172.25.170.139:53368 failed:
  java.io.EOFException |
  org.apache.activemq.broker.TransportCon
  nection.Transport |
ActiveMQ Transport:
  tcp:///172.25.170.139:53368@7862
5 2022-09-17 10:20:26,788 | WARN |
  Transport Connection to:
  tcp://172.25.170.139:53396 failed:
  java.io.EOFException |
  org.apache.activemq.broker.TransportCon
  nection.Transport |
ActiveMQ Transport:
  tcp:///172.25.170.139:53396@7862
6 2022-09-17 10:20:26,788 | WARN |
  Transport Connection to:
  tcp://172.25.170.139:53644 failed:
  java.io.EOFException |
  org.apache.activemq.broker.TransportCon
  nection.Transport |
ActiveMQ Transport:
  tcp:///172.25.170.139:53644@7862
7 2022-09-17 10:20:26,788 | WARN |
  Transport Connection to:
  tcp://172.25.170.139:53808 failed:
  java.io.EOFException |
  org.apache.activemq.broker.TransportCon
  nection.Transport |
ActiveMQ Transport:
  tcp:///172.25.170.139:53808@7862
8 2022-09-17 10:51:08,252 | WARN |
  Broker localhost not started so using
  degtlun5363 instead |
  org.apache.activemq.broker.BrokerRegistr
  y | qtp1659286984-90273
9 2022-09-17 16:13:48,573 | WARN |
  Broker localhost not started so using
  degtlun5363 instead |
  org.apache.activemq.broker.BrokerRegistr
  y | qtp1659286984-91217
10   2022-09-18 00:30:12,139 | INFO  |
  Apache ActiveMQ 5.15.8
(degtlun5363, ID:degtlun5363-39795-
  1662132976013-0:1) is shutting down |
  org.apache.activemq.broker.BrokerService
  | Thread-358
11   2022-09-18 00:30:13,344 | INFO  |
  Connector openwire stopped |
  org.apache.activemq.broker.TransportCon
  nector | Thread-358
12   2022-09-18 00:30:13,351 | INFO  |
  Connector amqp stopped |
  org.apache.activemq.broker.TransportCon
  nector | Thread-358
13   2022-09-18 00:30:13,352 | INFO  |
  Connector stomp stopped |
  org.apache.activemq.broker.TransportCon
  nector | Thread-358
14   2022-09-18 00:30:13,353 | INFO  |
  Connector mqtt stopped |
  org.apache.activemq.broker.TransportCon
  nector | Thread-358
15 2022-09-18 00:30:13,438 | INFO |
  Connector ws stopped |
  org.apache.activemq.broker.TransportCon
  nector | Thread-358
16   2022-09-18 00:30:13,462 | INFO  |
  PListStore:[/DBA/mule-
  home2/activemq515/data/degtlun5363/tm
  p_storage]stopped |
  org.apache.activemq.store.kahadb.plist.PL
  istStoreImpl | Thread-358
```

Fig: ActiveMQ logs used as an example

Grouping of log events

The next step after pre-processing is done is to group the log events together. This grouping is done so that the processor can distinguish between static and dynamic tokens with better accuracy when frequency analysis is applied to group members. The Log messages in the above ActiveMQ logs in lines 1-7 can be grouped together since all are related to transport connection. Similarly, 8 and 9 deal with broker localhost, and 11-15 show the connector-related logs. Line 16 will be put in a separate group since it is not matching with other log lines. The separate group then undergoes localized frequency analysis so that static tokens can be separated from dynamic tokens. Here, Date-Time, log level, and IP addresses are deemed dynamic, the rest all are considered as static tokens. The string matching technique is applied to separate log lines into tokens. For each log line, the number of words is counted in that logline, words being any set of characters separated by whitespace. Then, Only those tokens are identified which have alphanumeric characters, because others are most likely to be dynamic tokens. If there is an exact match between the number of tokens and the number of dynamic tokens, the log lines are grouped together. Now that we have different groups of logs, w e can now apply local frequency analysis to determine dynamic tokens and then generate log templates for each group.

Generation of tokens using local frequency analysis

In this step, the log group undergoes frequency analysis to determine the number of times a token has appeared in that group. A frequency threshold is set, and initially, all tokens are considered dynamic. If a token appears more times than the maximum frequency, it is then deemed static by the processor. To prevent bias originating from tokens occurring multiple times in the same log line, multiple tokens present in the same log lines are rejected. For example, the word Transport comes twice in each of lines 1-8, which is read-only once by the frequency analyzer.

Following is the table 1 of tokens generated after local frequency analysis is done in lines 1-8.

Log Pattern Recognition

After we have our tokens ready and the log lines are clustered, each cluster is assigned a pattern. For this process, we employ two techniques as given in LogMine [15], and both of them are discussed below.

Pattern Generation in Pairs

After the lothe pattern recognition algorithm is deployed only after clusteringge two log lines. If the two log lines are 100% match, the log lines are merged. We start processing the logs field by field once we have the log lines of similar length. We use the Smith-Waterman algorithm [28], one of many algorithms available for string alignment, to align two sequences of length l1 and l2 in $O(l1, l2)$ time, which causes time complexity of our Merge function to also be $O(l1, l2)$.

Sequential Pattern Generation

After we have determined the log lines' tokens, we are essentially going line-by-line and determining to which cluster that particular log line fits best. As you can imagine, ordering plays an important role in whether the log line will be put on a new or old cluster will be used. However, our end goal is to find log lines grouped in a different cluster, not how those clusters are arranged one after the other, which is not affected by the ordering of log lines. For this reason, we can use another technique, described in the next section.

Term	Frequency	Classification
2022-09-17	8	Dynamic Token
warn	8	static token
10:20:26788	7	Dynamic Token
transport	7	static token
connection	7	static token
to:	7	static token
failed:	7	static token
javaio.eofexception	7	static token
orgapache.activemq.broker.transportconnection.transport	7	static token
|activemq	7	static token
tcp://17225.170.139:53160	1	Dynamic Token
tcp:///17225.170.139:53160@7862	1	Dynamic Token
tcp://17225.170.139:53660	1	Dynamic Token
tcp:///17225.170.139:53660@7862	1	Dynamic Token
10:51:08252	1	Dynamic Token
broker	1	static token
localhost	1	static token
degtlun5363	1	static token
instead	1	static token
|orgapache.activemq.broker.brokerregistry	1	static token
qtp1659286984-90273	1	Dynamic Token

Table 1: Frequency Analysis of tokens

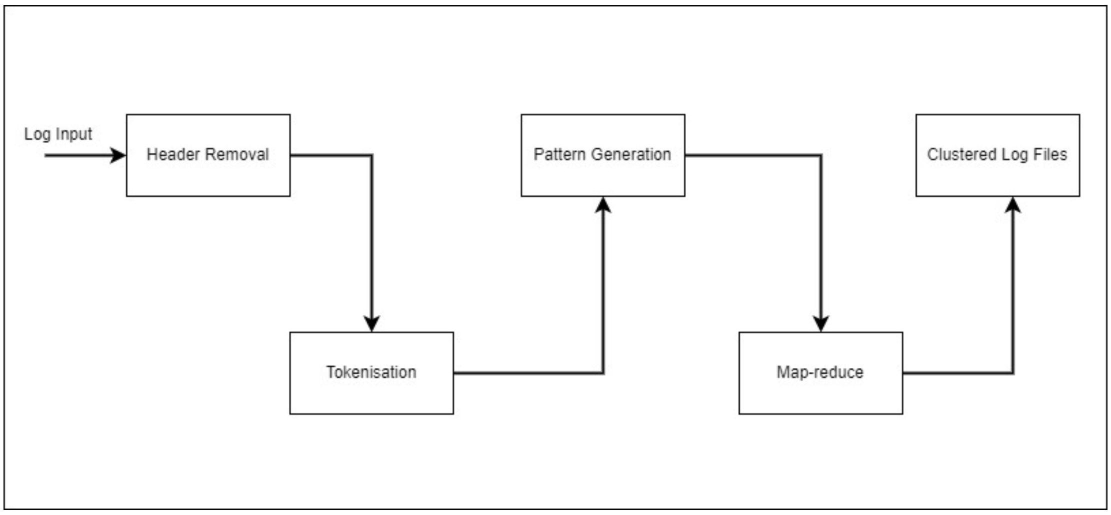

Figure 1: Flow Diagram of YALP

Improve Scalability via Map-Reduce Framework

As discussed in 3.4.2, the order of logs present in the file or the order of merging of log lines after tokenization is not important. So, in order to improve efficiency, we employ a map- reduce framework and parallelize this merging process. We know the cluster for every single log line since only after clustering the pattern recognition algorithm is deployed. In this process, whenever a new cluster is created, we assign a cluster number to it. In the map function, each log gets assigned a key-value pair. The cluster number created in the previous step is used as the key, and the value is the whole log line. If the map-reduce framework finds the same key, it will concatenate the line in reduce function. In the reduce function, the merging of two log lines in the same cluster happens. Consequently, in the final output, we get a cluster of logs. If we run this function on m machines, and the number of log lines is n, and l being the average number of tokens in each log line, the time complexity is $O(\frac{n}{m}.l^2)$

Evaluation

We evaluate YALP's accuracy and efficiency by parsing logs of 10 log datasets of the LogPai benchmark [34] available online. The datasets consist of a collection of log files from various systems, including Apache, HPC, HDFS.

The following questions needs to be answered to quantitatively evaluate YALP:

1. How precisely does log grouping occurs in YALP so that each of the millions of log gointo relevant cluster
2. How efficiently does YALP parse the cluster when compared to other parsing tools

Precision
For baseline, we will use the log clusters generated in OPTICS [6], and then define a baseline metric that will calculate how close the clustering of YALP is to that of OPTICS. Let the number of logs in the log lines be n, S = { l_1, l_2, l_3..., l_n }, M = { M_1, M_2, M_3..., M_n } be the cluster you get after running OPTICS, = { N_1, N_2, N_3..., N_n } be the set of cluster you get after running YALP. The agreement metric is $\frac{a}{}$, where a is the number of pairs of logs lines that are in S and in the same cluster set of *M*, and b is the number of pairs of logs lines that are in S and in the same cluster set of *N*.

Following is the table 2 which shows the accuracy comparison result of YALP with OPTICS and Logmine.

	Agreement Score	LogMine Accuracy	YALP Accuracy
D1	86%	100%	100%
D2	48%	73%	76%
D4	95%	96%	94%
D5	100%	98%	99%

Table 2: Comparing LogMine and YALP clustering accuracy with OPTICS

Efficiency
In section 3.4.2 and 3.4.3, we discussed the problem of parsing the line one by one and how we have increased the efficiency by adding map-reduce framework as defined in LogMine [15]. We take a dataset having 10 million log lines. and then run it through sequential log clustering and then via the map-reduce algorithm, initially working with 8 workers, each running 1 GB memory with a single core CPU. Map-reduce takes up to 5 times less time to cluster the logs than the sequential algorithm takes. This time can be further decreased by using more workers for clustering. The algorithm doesn't show much improvement for 10 million log lines when the number of workers is increased beyond 32.

Conclusion
We have presented YALP, which can take input from the stream, buffer, and file. It first generates tokens out of log lines, then classifies them into static and dynamic tokens. Once tokens are generated, the log lines are clustered using the map-reduce function to get log clusters. Since one log line is needed to go through processing only once, pattern generation is done in real-time.

Future work would be to generate Regex patterns on top of each cluster so that those Regex patterns can be used effectively to filter logs based on users' needs. A DevOps Engineer could generate a notification system that would be triggered when a certain amount of hits are encountered on that pattern. This would eliminate the need for the support services to look for errors inside the server, as the notification system can send the logs pertaining to that pattern directly.

References

[1] Elasticsearch: Store, search, and analyze. https://www.elastic.co/guide/index.html .

[2] Graylog: Data. insights. answers. https://www.graylog.org.

[3] Logentries by rapid. https://docs.logentries.com/.

[4] loggly. https://www.loggly.com/ultimate-guide/.

[5] Sumo logic: Making the world's apps reliable and secure. https://www.sumologic.com/.

[6] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.

[7] Spyros Blanas, Jignesh M Patel, Vuk Ercegovac, Jun Rao, Eugene J Shekita, and Yuanyuan Tian. A comparison of join algorithms for log processing in mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 975–986, 2010.

[8] Hetong Dai, Heng Li, Che Shao Chen, Weiyi Shang, and Tse-Hsun Chen. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Transactions on Software Engineering*, 2020.

[9] Chen Ding and Jin Zhou. Log-based indexing to improve web site search. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 829–833, 2007.

[10] Ignacio Cabrera Dominique Karg, Julio Casal and Alberto Román. Open source security information management. https://en.wikipedia.org/wiki/OSSIM, 2022.

[11] Min Du and Feifei Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864. IEEE, 2016.

[12] Diana El-Masri, Fabio Petrillo, Yann-Gaël Guéhéneuc, Abdelwahab Hamou-Lhadj, and Anas Bouziane. A systematic literature review on automated log abstraction techniques. *Information and Software Technology*, 122:106276, 2020.

[13] Mirghani A Eltahir and Anour FA Dafa-Alla. Extracting knowledge from web server logs using web usage mining. In *2013 INTERNATIONAL CONFERENCE ON COMPUTING, ELECTRICAL AND ELECTRONIC ENGINEERING (ICCEEE)*, pages 413–417. IEEE, 2013.

[14] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*, pages 149–158. IEEE, 2009.

[15] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. Logmine: Fast pattern recognition for log analytics.

In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1573–1582, 2016.

[16] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. An evaluation study on log parsing and its use in log mining. In *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 654–661. IEEE, 2016.

[17]Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, pages 33–40. IEEE, 2017.

[18] Zhen Ming Jiang, Ahmed E Hassan, Gilbert Hamann, and Parminder Flora. An automated approach for abstracting execution logs to execution events. *Journal of Software Maintenance and Evolution: Research and Practice*, 20(4):249–267, 2008.

[19] Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. Towards an nlp-based log template generation algorithm for system log analysis. In *Proceedings of The Ninth International Conference on Future Internet Technologies*, pages 1–4, 2014.

[20] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[21] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *AcM sIGMoD record*, 40(4):11–20, 2012.

[22] Adetokunbo Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. A lightweight algorithm for message type extraction in system application logs. *IEEE Transactions on Knowledge and Data Engineering*, 24(11):1921–1936, 2011.

[23] Erik Swan Michael Baum, Rob Das. Splunk. https://www.splunk.com/en_us/solutions/ it-modernization.html, 2022.

[24] X Ning and G Jiang. Hlaer: A system for heterogeneous log analysis, 2014. In *SDM Workshop on Heterogeneous Learning*, 2014.

[25] Raghunath Rajachandrasekar, Xavier Besseron, and Dhabaleswar K Panda. Monitoring and predicting hardware failures in hpc clusters with ftb-ipmi. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 1136–1143. IEEE, 2012.

[26] K Sudheer Reddy, G Partha Saradhi Varma, and I Ramesh Babu. Preprocessing the web server logs: an illustrative approach for effective usage mining. *ACM SIGSOFT Software Engineering Notes*, 37(3):1–5, 2012.

[27] Issam Sedki, Abdelwahab Hamou-Lhadj, Otmane Ait-Mohamed, and Mohammed A Shehab. An effective approach for parsing large log files. In *ICSME*, 2022.

[28] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences.*Journal of molecular biology*, 147(1):195–197, 1981.

[29] Stefan Thaler, Vlado Menkonvski, and Milan Petkovic. Towards a neural language model for signature extraction from forensic logs. In *2017 5th International Symposium on Digital Forensic and Security (ISDFS)*, pages 1–6. IEEE, 2017.

[30] Risto 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)*, pages 119–126. Ieee, 2003.

[31]Risto Vaarandi. Mining event logs with slct and loghound. In *NOMS 2008-2008 IEEE Network*