ISSN: 2633-4828 Vol. 5 No. S4, (July – August 2023)

International Journal of Applied Engineering & Technology

APPLICATION SPECIFIC LOG PARSER

Rijul Saxena [1], Rohan Sabu Thomas [2], Rohan Lakshman Shetty [3], Sankalp Naveenachandra Kulkarni [4], Avinash Tiwari [5], Dr. Nagegowda K.S. [6], Shibu Prasad [7], Bikee Bihari [8]

[1, 2, 3, 4] PES University, Bengaluru, India.

[6] Professor, PES University, Bengaluru, India.

[5, 7, 8] pCloudy.com, Dublin, California

[1] rijul.saxena@gmail.com, [2] rohan.st2002@gmail.com, [3] rohanshettu1@gmail.com, [4] sankalpkulk06@gmail.com, [5] avinash.tiwari@sstsinc.com, [6] nagegowda@pes.edu, [7] shibu.prasad@sstsinc.com, [8] bikee.bihari@sstsinc.com

Abstract

A log in software engineering, however, is a document produced from various frameworks, services, and applications. This study introduces a new online tool called Android Log Parser Utility that is designed to parse online logs for Android devices. It works in real time streaming mode allowing the filtering of dynamic logs to only present those related to a specific application. Finally, there are several filter tools available such as focus on keywords and whole search filters to enable fast location of these logs with ease. Furthermore, this utility tool can make suggestions for repairing faulty log lines as well as proposing corrective actions which streamline debugging process leading to better user experience. The main aim of this multi-faceted tool is to provide our customers with an effective solution for dealing with logging data within the context of android app development.

***Keywords**: Android logs, Application specific logs, Keyword search, Log parsing, Log mining, Streaming data.*

INTRODUCTION

It is not arguable that the mobile application landscape is changing very fast, which has captivated many researchers [24]. With modernity embraced by people, millions of individuals depend on mobile apps for their daily activities such as social media networking (e.g., Facebook, Twitter, LinkedIn, and Instagram), online banking, and emailing [25]. Many new applications are developed each year with forecasts showing a significant global download volume and revenue [26]. Therefore, it is necessary to test extensively mobile applications to create high quality ones that meet user expectations. However, studies indicate that developers often emphasize backend functionality rather than user experience. An app's reputation can be shaped through feedback from its users which will in turn determine whether an app succeeds or fails [27]. When faced with any glitches in an application, users lose interest quickly making them abandon after few trials.

In this landscape, logs play a pivotal role in the comprehensive documentation of runtime data pertaining to software systems. The wealth of information encapsulated within logs serves as a valuable resource, empowering system developers and operators to vigilantly observe the dynamic runtime characteristics of their systems. Additionally, logs facilitate the proficient identification and troubleshooting of system anomalies occurring within production environments.

The number of logs being generated is on the increase due to rapid system expansion and rising complexity of contemporary systems, with a 50 gigabytes hourly rise (equal to about 120-200 million lines) [1]. This makes manual log analysis methods rather labor-intensive and error-prone. In response to this challenge, there has been a recent surge in attempts to automate log analysis using data mining techniques. These include anomaly detection [2], [3], [4], program verification [5], [6], problem diagnosis [7], [8] and security assurance [9], [10]. Nevertheless, raw log messages normally lack structure as developers tend to employ free-form text for logging purposes that emphasize convenience, and flexibility. Log parsing is the first step in aiding automated analysis of unstructured logs whereby unstructured raw log messages are transformed into a sequence of structured events.

Generally, a log message, as demonstrated in the ensuing instance, serves to document a particular system event, comprising a predefined set of attributes, namely: timestamp (denoting the moment of occurrence of the event), verbosity level (signifying the severity level of the event, such as "INFO" denoted as 'I/" in android logs), process

ID (the process that created this message), and the unaltered message content (detailing the transpired events during system operation).

09-25 11:33:42.271 I/Launcher.Model(1797): package changed received : com.myntra.android

The above example demonstrates the idea of splitting a raw log message into two parts namely: "constant part" and "variable part" and shows the most important aspect of log analysis. The constant part is unchanging in all log messages; it always includes clues about the nature of the events that are being registered to (via verbosity level). This fixed portion resembles a fingerprint which helps us know what kind of event it is. On the other hand, the variable part is like the chameleon in log message. It contains real-time information that varies greatly from one logging entry to another. This flexible character of a log message often involves key runtime details like changing states, parameters or others relevant information tied to an event at stake. For instance, it could tell us about new process start-ups, unexpected app termination as well as important system stats. The main objective for parsing logs is thus automating extraction of meaningful events from raw logs data. Attaining this target necessitates effective differentiation as well as separation between constant and variable portions in each logging message. In so doing, developers will be able to see how their applications work and identify problems quickly to diagnose them easily.

Many recent studies have proposed various data-driven approaches for automated log parsing to achieve this goal such as SLCT [13], IPLoM [14], LKE [3] and LogSig [15] which utilize historical log data for training statistical models for event extraction. However, despite its importance in the field of computer science, there is a lack of comprehensive evaluations on how effective and efficient these automated techniques are. Except SLCT [13] that was introduced over a decade ago, there are no readily available implementations of log parsers. And even with commercial log management solutions like Splunk [16] and Logstash [17], users need to provide intricate configurations with custom rules to parse their logs.

Another pertinent issue is the prevailing lack of awareness among developers regarding the effectiveness of their in-house log parsing implementations in comparison to alternative methods. Additionally, the potential influence of log parsing on subsequent log mining tasks often goes unnoticed, further underscoring the need for systematic evaluations and readily accessible log parsing tools to streamline and enhance the log mining process.

Given that this product represents a pioneering advancement within the industry, it is important to note that direct head-to-head comparisons with existing log parsers currently available in the market have not been feasible. Consequently, our evaluation of the product's accuracy has primarily relied on customer feedback, which has, overall, been positive and encouraging.

READED WORK

In the last few years, there has been a deluge of studies focusing on log analysis research by both academics and industry practitioners. This increased attention is due to the need for real-time understanding of intricate systems. El-Masri et al. [16] carried out an extensive survey on log parsing techniques, and proposed a quality model for their classification. To support their categorization, they extensively examined 17 different log parsing tools.

Log parsing tools are available in various forms that include rule-based approaches, frequent token mining, natural language processing and classification/clustering methods. This exposition will probe into fundamental tenets of each approach before concluding with a comprehensive comparison between our log parser and these traditional approaches.

Vaarandi et al. [17] and [18] introduced SLCT which is an acronym for Simple Logfile Clustering Tool. It employs a density-based clustering algorithm to detect dynamic tokens. Also, it utilizes frequency analysis to implement it. Lastly, LogCluster [19], an improved version of SLCT was built where terms were extracted from frequently logged messages as tuples before clustering them into related logs.

IPLOM (Iterative Partitioning Log Mining) method by Makanju et al. [20] uses a heuristic-based hierarchical clustering algorithm in its log identification phase. It starts by splitting logs according to their event size and then further splits the partitions based on the partition that shares the highest number of terms together. In another

approach called LKE (Log Key Extraction), Fu et al. [21] group log events with weighted edit distance and give more weight to those whose terms are occurring at the beginning of the log events. The process continues until all log clusters contain equal log keys hence most common parts are identified as patterns for event types definition.

The Drain method as described in Reference [22] consists of five main steps. It uses regular expressions to preprocess raw log messages and identifies simple dynamic tokens by grouping them, as we do with our log parser. Then a parse tree is built for each log event based on the number of its tokens assuming that tokens appearing first are generally static. To distinguish between different groups of logs, the method uses a similarity metric which compares leaf nodes with event types.

Additionally, in paper [23], there was presented Spell (Streaming Parser for Event Logs using an LCS) that had been developed to convert log messages into separate events where shared logging statement types are denoted by longest common sequence.

METHODOLOGY

Our log parser broadly consists of the following steps: pre-processing, segregation of logs based on timestamp and verbosity level, displaying statistics on the website. In the subsequent sections, we provide a more comprehensive breakdown of each step in our methodology. To elucidate our approach further, we utilize a set of sample log events derived from a real time log capture of a running application.

A. Pre-processing

The pre-processing phase of log data initiation commences by initially determining the point of origin for an application launch, as indicated by the application package name, a piece of information provided to us in advance by the consumer prior to the commencement of testing. The program remains in a waiting state until it detects the initial occurrence of the application's launch. When a log entry that contains the specified package name is found, it extracts out the associated process ID and then the program keeps track of only those log entries which belong to a particular process ID. Furthermore, when the program enters running mode indicating that it has successfully identified the parent process ID linked with the relevant application package name, it continues to be on alert for any possible child processes that can be created by this parent process. The generated child process IDs are also monitored and logged together with other remaining data in our system. This whole procedure uses standard regular expressions, which facilitate better matching patterns and identifying important log entries. In addition, to facilitate the organization and segmentation of relevant log messages, we have implemented a 0.5-second interval for log subdivision. This approach not only aids in the systematic grouping of logs but also plays a pivotal role in the subsequent grouping of logs, as elaborated upon in the ensuing subsection.

B. Segregation of Logs Based on Timestamp and Verbosity Level

In order to visually represent our data and facilitate sorting by verbosity level, we have structured our stored data in a manner that enhances accessibility for our frontend. To achieve this, we initially categorize all pertinent logs into 0.5-second time intervals, as previously mentioned. Within each of these 0.5-second intervals, we maintain a record of the total count of logs contained therein.

Furthermore, we carry out a secondary categorization, classifying the logs based on their verbosity levels (e.g., "I/" for information, "V/" for verbose). In addition to the standard log levels, we have introduced an extra log level denoted as "FATAL." This additional category is dedicated to capturing and segregating fatal errors independently, as we believe it holds heightened significance for our customers. Within the division by verbosity level, we store both the frequency of logs corresponding to the respective log level and the actual log messages that pertain to a particular verbosity level. This organization and structuring of data aim to facilitate an efficient and user-friendly representation for our frontend visualization.

C. Displaying Statistics on the Website

To deliver a smooth experience to the users, we have used Node.js and React to serve the webpage content. The main page is a dynamic line chart that shows how many log lines are there for each time they appear in the log file. This graph takes its data from Redis, where we have stored all our relevant data. Moreover, the graph contains

filtering functionality through which users can change what they see by ordering log entries based on their verbosity level and timestamp.

The interface of our website includes not only a line chart but also a pie chart that demonstrates the distribution of different modes of verbosity as percentages of total log lines. This pictorial view offers an overall picture of how logs are built up.

We also introduced an easy-to-use search bar where consumers can enter keywords to narrow down their results to specific log entries. For instance, whenever such searches are performed on the backend side after hitting the search button, related lines will be displayed. Internally this is achieved using general regular expressions for the filters.

Our platform comes with an insight page for users who want to get deeper into a particular log entry. It gives more elaborate explanations of the chosen log line and, when need be, proposes solution links for bugs. To generate such links, a Selenium script in the background runs and searches the web for possible fixes. By scraping related website links for fixes, this script makes sure that users can access all information and advice they need.

Fig. 4.1. Website Front Page

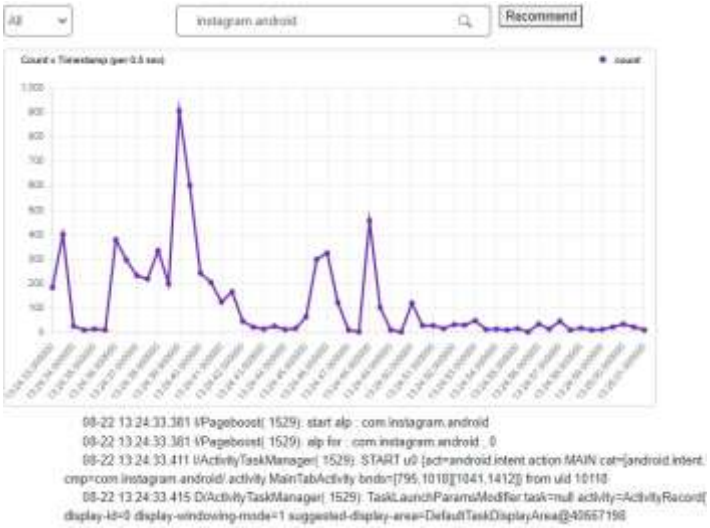

Fig. 4.2 Keyword Filtering

RESULTS

In this section, we present our log parsing utility application. Since our product is pioneering in the industry, traditional metrics such as accuracy or Mean Absolute Error are not applicable. Therefore, we rely solely on customer feedback, as mentioned earlier.

To illustrate our findings, we conducted a 5-second log capture of the Instagram application being launched and interacted with on an Android device. Each aspect is detailed as follows:

A. Front Page Interaction:
Upon clicking the link from the main website, the following features are observed (See Fig. 4.1):

1. A line graph depicting lines consumed per 0.5 seconds with a verbosity level filter.
2. A pie chart displaying verbosity level percentage values.

B. Keyword Filtering:
We conducted a simple search using the application package name, in this case, the package name is "com.instagram.android" (See Fig. 4.2).

C. Solution Links Page:
A random logline was selected and clicked on to navigate to the Solution Links page (See Fig. 4.3).

Fig. 4.3. Solution Links Page

Fig. 4.4. Redis Data Store

D. Miscellaneous Observations:
1) Redis Store:
Our Redis data store is shown in Fig. 4.4. We store the relevant log data, previous keyword search results, and data for our graph generation.

2) Kafka Implementation:
We elaborate on the integration of Kafka within our system, detailing the live capture performed by our Kafka script during the test phase. It is noteworthy that only application-specific log lines are fed into the Redis database. Additionally, we have also shown the matches our regex pattern finds (See Fig. 4.5).

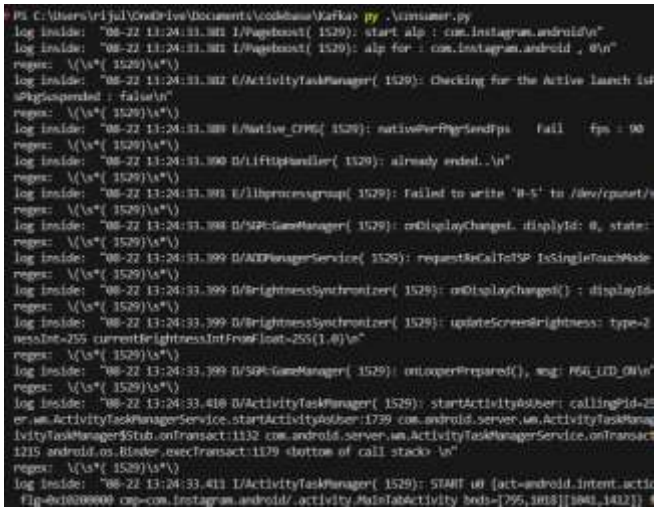

Fig. 4.5. Kafka consumer script

CONCLUSIONS

The utilization of log files is a treasure trove of invaluable data which can improve the efficiency of follow up testing stages. The Alpha Testing stage for our product has shown promising results and motivated us to deliver a scalable, robust platform for our customers as well.

Furthermore, our product currently can only parse android log data. Researchers are encouraged to explore and develop a working solution for iOS log data as well.

We extend our gratitude to Mr. Avinash Tiwari, Co-Founder of pCloudy.com, for affording us the valuable opportunity to embark on the creation of this product and for his continuous guidance and provision of essential resources and knowledge throughout the entire development cycle. Additionally, we express our appreciation to our mentor, Dr. Nagegowda K S, for his invaluable guidance in shaping and advancing the design and development processes of the product.

REFERENCES

[1] H. Mi, H. Wang, Y. Zhou, R. Lyu, and H. Cai, “Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems,” IEEE Transactions on Parallel and Distributed Systems, vol. 24, pp. 1245–1255, 2013.

[2] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordon, “Detecting largescale system problems by mining console logs,” in SOSP’09: Proc. of the ACM Symposium on Operating Systems Principles, 2009.

[3] Q. Fu, J. Lou, Y. Wang, and J. Li, “Execution anomaly detection in distributed systems through unstructured log analysis,” in ICDM’09: Proc. of International Conference on Data Mining, 2009.

[4] A. Makanju, A. Zincir-Heywood, and E. Milios, “Fast entropy based alert detection in super computer logs,” in DSN-W’10: Proc. of International Conference on Dependable Systems and Networks Workshops, 2010, pp. 52–58.

[5] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. Ernst, “Leveraging existing instrumentation to automatically infer invariantconstrained models,” in ESEC/FSE’11: Proc. of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, 2011.

[6] W. Shang, Z. Jiang, H. Hemmati, B. Adams, A. Hassan, and P. Martin, “Assisting developers of big data analytics applications when deploying on hadoop clouds,” in ICSE’13: Proc. of the 35th International Conference on Software Engineering, 2013, pp. 402–411.

[7] D. Yuan, S. Park, P. Huang, Y. Liu, M. Lee, X. Tang, Y. Zhou, and S. Savage, "Be conservative: enhancing failure diagnosis with proactive logging," in OSDI'12: Proc. of the 10th USENIX Conference on Operating Systems Design and Implementation, 2012, pp. 293–306.

[8] K. Nagaraj, C. Killian, and J. Neville, "structured comparative analysis of systems logs to diagnose performance problems," in NSDI'12: Proc. of the 9th USENIX conference on Networked Systems Design and Implementation, 2012.

[9] A. Oprea, Z. Li, T. Yen, S. Chin, and S. Alrwais, "Dectection of earlystage enterprise infection by mining large-scale log data," in DSN'15, 2015.

[10] Z. Gu, K. Pei, Q. Wang, L. Si, X. Zhang, and D. Xu, "Leaps: Detecting camouflaged attacks with statistical learning guided by program analysis," in DSN'15, 2015.

[11] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in IPOM'03: Proc. of the 3rd Workshop on IP Operations and Management, 2003.

[12] A. Makanju, A. Zincir-Heywood, and E. Milios, "Clustering event logs using iterative partitioning," in KDD'09: Proc. of International Conference on Knowledge Discovery and Data Mining, 2009.

[13] L. Tang, T. Li, and C. Perng, "LogSig: generating system events from raw textual logs," in CIKM'11: Proc. of ACM International Conference on Information and Knowledge Management, 2011, pp. 785–794.

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

[15] Logstash. [Online]. Available: http://logstash.net

[16] D. El-Masri, P. Fabio, Y.-G. Gueheneuc, A. Hamou-Lhadj, and A. Bouziane, "A systematic literature review on automated log abstraction techniques," Information and Software Technology, vol. 122, pp. 106–276, 02 2020.

[17] R. Vaarandi, "Mining event logs with slct and loghound," in Network Operations and Management Symposium, 2008. NOMS 2008. IEEE. IEEE, 2008, pp. 1071–1074.

[18] ——, "A data clustering algorithm for mining patterns from event logs," in IP Operations & Management, 2003. (IPOM 2003). 3rd IEEE Workshop on. IEEE, 2003, pp. 119–126.

[19] 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), 2015, pp. 1–7.

[20] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 11, pp. 1921–1936, 2012.

[21] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on. IEEE, 2009, pp. 149–158.

[22] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in Web Services (ICWS), 2017 IEEE International Conference on. IEEE, 2017, pp. 33–40.

[23] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in Data Mining (ICDM), 2016 IEEE 16th International Conference on. IEEE, 2016, pp. 859–864.

[24] Dongsong, Z. and Adipat, B. (2005). "Challenges Metho dologies, and Issues in the Usability Testing of Mobile Applications", International Journal of Human Computer Interaction, Vol. 18, 3.

[25] Jacob, J.and Tharakan, M. (2012). "Roadblocks and their workaround, while testing Mobile Applications".The Magazine for Professional Testers. P8-16, Vol 19. [Online] Available at: http://www.testingexperience.com/ [Accessed 23rd September 2014].

[26] Muccini, H., Di Francesco, A. and Esposito, P. (2012), "Software testing of mobile applications: Challenges and future research directions," Automation of Software Test (AST), 2012 7th International Workshop on, vol., no., pp.29,35 [online] Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6228987&isnumber=6228973 [Accessed 23rd September 2012].

[27] Haller, K. (2013). "Mobile Testing". ACM SIGSOFT Software Engineering Notes, 38, 6, 1-8. [Online] Available at: http://doi.acm.org/10.1145/2532780.2532813 (November 3rd, 2014).

[28] P. He, J. Zhu, S. He, J. Li and M. R. Lyu, "An Evaluation Study on Log Parsing and Its Use in Log Mining," 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Toulouse, France, 2016, pp. 654-661, doi: 10.1109/DSN.2016.66.