2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)

An Empirical Study On Leveraging Logs For Debugging Production Failures

An Ran Chen
Department of Computer Science and Software Engineering
Concordia Univerity
Montreal, Canada
anr_chen@encs.concordia.ca

***Abstract*—In modern software development, maintenance is one of the most expensive processes. When end-users encounter software defects, they report the bug to developers by specifying the expected behavior and error messages (e.g., log message). Then, they wait for a bug fix from the developers. However, on the developers' side, it can be very challenging and expensive to debug the problem. To fix the bugs, developers often have to play the role of detectives: seeking clues in the user-reported logs files or stack trace in a snapshot of specific system execution. This debugging process may take several hours or even days.**

In this paper, we first look at the usefulness of the user-reported logs. Then, we propose an automated approach to assist the debugging process by reconstructing the execution path. Through the analysis, our investigation shows that 31% of the time, developer further requests logs from the reporter. Moreover, our preliminary results show that the reconducted path illustrates the user's execution. We believe that our approach proposes a novel solution in debugging production failures.

***Index Terms*—Mining Software Repository Data, Events Log, Static Analysis, Production Errors**

I. Introduction

In the current state-of-the-arts, there are a number of studies on the challenges of debugging production failures from bug reports. Saha et al. [11] based their approach on information retrieval techniques to improve bug localization. Mills et al. [5] tackled the challenge of analyzing bug report vocabularies to facilitate the debugging process. Among those researches, logs, which contain tremendous values of information, are rarely used.

Ding et al. found that "*for a majority (84%) of the failures, all of their triggering events are logged.*" [7]. However, one of the major challenges in diagnosing bug reports relies on how to effectively analyze the provided logs. In modern software debugging, developers often uses *grep* or more sophisticated search utilities to uncover the related context, but the debugging process is still challenging [1]. It is time consuming, and manual inspection quickly becomes problematic when dealing with large-scale systems. In short, considering the complexity of modern softwares, it is difficult for users to point out what went wrong in the execution. This make the debugging task challenging on the developer's end, and logs are the primary source of debugging hints that developer refers to.

In this paper, we start by analyzing the presence of logs in bug reports. Then, we evaluate the usefulness of user-reported logs by comparing the resolution time of bug report with logs with those without logs. Finally, we propose an automated approach that utilizes logs to reconstruct the execution paths.

Section II provides an overview of our experimental setup, and a preliminary study on the presence of logs in bug reports across different Apache Software Foundation open-source systems. Section III discusses the main research questions of this study. Section IV concludes the paper.

II. Experimental Setup

For our case study, we have selected six open-source Java systems: Hadoop, Hive, Camel, Storm, ActiveMQ and Zookeeper. We select projects from Apache Software Foundation for two reasons: 1) we have access to most of its code base, version control and issue tracker; 2) the development process of those projects is stable, and many of them are actively maintained. The studied systems vary from virtual machine deployment to data warehouse. Table I shows the size of those projects in terms of lines of code (LOC) (computed by cloc [8]).

Presence of logs in bug report: a preliminary study. To come up with these five Java projects, we have done a preliminary study across 16 Apache Software Foundation open-source projects. As our empirical study focuses on the effect of having log messages when debugging, we decide to proceed with five projects that contain the most bug reports with logs. To retrieve the bug reports, we built a web application that performs REST API calls to Apache Jira repositories [9]. The interface of our tool collects bug reports based on user's selection criteria. For this study, we extracted the bug reports based on the criteria of a prior study (Yuan et al. [7]).

We present the result of this preliminary study as follows. We calculated the percentage of bug report with log over the total percentage of bug reports. Despite the fact that only a small portion of the reporters attach logs in bug reports (varying between 0.21% and 8.45%), this still leaves us a reasonable amount of bug reports to study.

III. Research Questions

We now present our research questions into two parts. First, we want to know how helpful are user-reported logs in the event of production failures. Furthermore, we present our approach, which reconstructs the execution path based on logs, so developers can easily debug and view the failures in

TABLE I
METRICS COMPARAISON ACROSS STUDIED SYSTEM WITH BRWL AS BUG REPORT WITH LOGS AND BRNL AS BUG REPORT WITHOUT LOGS

Project	LOC	BRWL	Total BR	BRWL median resolution time	BRNL median resolution time
ActiveMQ	414k	201	4,590	2085	247
Camel	1,123k	9	4,275	137	22
Hadoop	1,691k	781	20,893	287	242
Hive	1,343k	208	11,479	188	160
Storm	278k	116	1,507	136	220
Zookeeper	92k	151	1,788	2081	453

the context of its execution. Finally, we evaluate our approach based on our preliminary results.

RQ1: Can user-reported logs help to debug production failures?

Motivation. Debugging for production failure can be challenging with the complexity of modern softwares. On one hand, when debugging from a bug report, it is challenging for developer to understand the specific unexpected behavior. Although there might be contextual hints that suggest potentially problematic areas, the nature of the failure remains unknown. One fundamental problem is not been able to reproduce the failure based on the provided information. On the other hand, it is difficult for the users to report specific usage details (e.g., underlying environment or contextual parameters used), and even harder to provide scripts to reproduction. Therefore, we want to investigate the usefulness of user-reported logs in the event of production failures.

Approach. We statistically compare the resolution time of the bug reports with logs (BRWL) and the bug reports without log (BRNL). We demonstrate with Wilcoxon rank-sum test that the resolution time of BRWL and BRNL are statistical significance. Furthermore, we also look at where were the logs attached in bug reports. If the logs were found in the Comments section, it is requested by the developers.

Preliminary Results. Table I shows the median resolution time of bug report with logs and without log. We use Wilcoxon rank-sum test to analyse if logs statistically improve the bug report resolution time. The null hypothesis is that there is no statistically significant difference in the value of resolution time between bug reports with logs and the ones without logs. We evaluate our null hypothesis for p-value$<$0.05 at a confidence level of 95%. The analysis across all studied systems outputs a p-value much smaller than 0.05, which indicates there is statistically significant difference in resolution time. The results of median resolution time shows that bug reports with logs take more time to resolve. After initiation investigation, we find out developer requests logs when the failure is more complex to debug or labelled as *Critical* failure. We also observe that 31% of the logs were found in Comments section of the bug report. Those logs are attached upon developer's request to better understand the failure.

RQ2: Can we utilize user-provided logs to assist debugging?

Motivation. In this RQ, we intend to utilize the logs to reproduce the execution path that leads to the reported failure. The outcome will assist developers to debug and visualize the possible failure locations. We implemented our approach into an execution path reconstruction tool called LogMap.

Approach. LogMap first identifies and retrieves log messages from bug reports using regular expression. Then, the tool leverages static analysis technique to map each log message to its corresponding logging line in source code. LogMap inspects user-reported log messages and program source code to determine which parts of the code were called during users' execution. It matches the key diagnostic information from log messages to static text located inside logging statements. The output of this static analysis are a collection of mapping between log messages and log statements implemented inside the source code. Finally, based on on this log message mapping, we traverse through the logging lines to derive a list of potential code paths. Specifically, LogMap utilizes JavaParser [13] to build an abstract syntax tree (AST) representation from every Java file. It traverses through each method declaration, monitoring all method calls inside the declaration to derive a call graph of the paths. Specifically, LogMap follows the implementation of Depth First Traversal directed graph [12]. LogMap analyzes the execution path by pairs of log messages - one as the source, the other as destination. After knowing the location of these log messages, it derives the execution paths that the program might have taken by traversing the graph to find a path between the source and destination vertex.

Preliminary Results. Using LogMap, we reproduced the execution path from 20 bug reports. We manually verified the execution path from the source code, and our approach was able to find the correct path which connect the log messages. Moreover, we confirmed the reproduced execution paths illustrate the context in which the user has described.

In the continuation of our research, we intend to analyze a research question as follows to better evaluate our approach. RQ3: How far away is the problematic code from the production logging statements? (i.e., in terms of the number of non-basic code blocks between the log execution paths and changed code). If we could justify that the number of non-basic code blocks are not significant, it will help us to evaluate the effectiveness of our tool. We also plan to implement an IDE plugin that assist developers with debugging by visualizing the reproduced execution. We plan to conduct a user study to evaluate our approach and the implemented plugin.

IV. CONCLUSION

Our work justifies the helpfulness of user-reported logs and provides an approach that fully utilizes these logs to reconstruct the execution path. Our main evaluation is still in progress, we have high expectations in our future work. Based on the preliminary results, we will examine the overlaps between the bug fixing commits and the identified execution path. The outcome will confirm our approach and reveal the advantages of such static analysis. We can see our tool to be complementary to many of state-of-the-art log analysis tools, such as Loggly or Logstash, and it could easily be integrated as an IDE plugin.

REFERENCES

[1] Tse-Hsun Chen, Mark D. Syer, Weiyi Shang, Zhen Ming Jiang, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. *Analytics-Driven Load Testing: An Industrial Experience Report on Load Testing of Large-Scale Systems.*. In ICSE-SEIP '17; 243-252.

[2] Saha RK, Lease M, Khurshid S, Perry DE. *Improving bug localization using structured information retrieval.*. In ASE, 2013; 345–355.

[3] Min D, Feifei L, Guineng Z, Vivek S. *DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning.*. In ACM SIGSAC, Conference on Computer and Communications Security(CCS), 2017; 1285-1298.

[4] Tse-Hsun Chen, Stephen W. Thomas, Ahmed E. Hassan *A Survey on the Use of Topic Models when Mining Software Repositories*. Empirical Software Engineering, 2016, Volume 21, Number 5, Page 1843

[5] Chris Mills, Jevgenija Pantiuchina, Esteban Parra, Gabriele Bavota, Sonia Haiduc *Are Bug Reports Enough for Text Retrieval-based Bug Localization?*. ICSE'18 Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, Pages 248-249

[6] Boyuan Chen, Zhen Ming (Jack) Jiang *Characterizing logging practices in Java-based open source software projects – a replication study in Apache Software Foundation*. Empirical Software Engineering, 2017, Volume 22, Number 1, Page 330-374.

[7] Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U. Jain, Michael Stumm *Simple Testing Can Prevent Most Critical Failures*. OSDI'14 Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation, Pages 249-265
Tools and Benchmarks for Automated Log Parsing

[8] cloc. 2018. Count Lines of Code (cloc). (2019) https://github.com/AlDanial/cloc

[9] Apache Software Foundation. 2018. Apache's JIRA issue tracker. (2019) https://issues.apache.org/jira/secure/Dashboard.jspa

[10] S. Nakagawa and I. C. Cuthill. *Effect size, confidence interval and statistical significance: a practical guide for biologists*. Biological Reviews, 82:591–605, 2007.

[11] Ripon K. Saha, Matthew Lease, Sarfraz Khurshid, Dewayne E. Perry *Improving Bug Localization using Structured Information Retrieval* ICCIT'18 International Conference on Computer and Information Technology (ICCIT)

[12] GeeksforGeeks *Depth First Searçh* https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

[13] JavaParser *JavaParser - For Processing Java Code* http://javaparser.org/