Multi-source log clustering in distributed systems

Jackson Raffety[1], Brooklynn Stone[1], Jan Svacina[1], Connor Woodahl[1], Tomas Cerny[1], and Pavel Tisnovsky[2]

[1] Baylor University, Waco TX 76701, USA
[2] Red Hat, FBC II Purkyňova 97b, 612 00 Brno, Czech Republic
tomas_cerny@baylor.edu

Abstract. Distributed systems are seeing wider use as software becomes more complex and cloud systems increase in popularity. Preforming anomaly detection and other log analysis procedures on distributed systems have not seen much research. To this end, we propose a simple and generic method of clustering log statements from separate log files to perform future log analysis. We identify variable components of log statements and find matches of these variables between the sources. After scoring the variables, we select the one with the highest score to be the clustering basis. We performed a case study of our method on the two open-source projects, to which we found success in the results of our method and created an open-source project log-matcher.

Keywords: distributed system · log clustering · multi-source logs.

1 Introduction

A distributed system is a complex set of machines that exchanges information and provides computational resources [14]. These systems got significant attention in recent years due to the increasing demand for conducting large computations that a single machine cannot manage. The distributed nature of the systems makes it particularly challenging to analyze them for anomalies, security intrusions, and attack patterns [14]. Traditionally, researchers proposed methods for detecting these errors by analyzing a log file, which are messages that the system outputs during its executions [9]. Nowadays, distributed systems became heterogeneous, with each machine producing differently structured messages [14]. The heterogeneity makes analysis of multiple log files challenging as there are no obvious elements that would connect events from machines across the distributed system [16,26]. Analyzing and connecting events across the distributed system is critical for finding anomalies or intrusions [6]. We propose a new generic method for finding common attributes in the log messages of any distributed system.

Our method proposes a simple way to cluster log statements from separate files with each other. Each log line contains static elements-which are the same text in all log statements (ie. in "`time=1:34:00`", time is the static element)- and variable elements-which dynamically change throughout the log statements

(ie. in "`orgID=148253052`", "`148253052`" is the variable element). We begin by extracting the variable elements from each line of the log files and organize them by their regular expression (an example would be an element "IP Address" with a regular expression of "`1(\d{1,3}\.){4}`" and a variable element of "`127.0.0.1.`"). We then cross-compare the variable elements between the two log files, only considering those whose regular expressions are the same. We record variable elements mapping to each static element across the files and find a match between files - if two files share a common variable element mapping to a static element. We then send the plain text static elements, and their match counts to a scoring function based on their uniqueness, frequency, and commonality from our matches. We output two static elements with the highest score as the variable component for which the two separate log files can be clustered.

We tested our proposed method on a case study using log files provided by Red Hat from their two open-source projects called *ccx-data-pipeline* (pipeline) [20] and *ccx-data-aggregator* (aggregator) [19]. With the highest score, we detected static element `clusterID` and used the values to successfully cluster statements across both systems. We manually validated our results with the quality assurance team at Red Hat for its correctness.

This paper is organized as follows. Section 2 provides background and discusses related works. Section 3 describes our method. We share a case study in Section 4 and conclude the paper in Section 5.

2 Background and Related Work

We aim to analyze and match logs originating from an overall distributed system. Log are output messages generated by the system during a process to describe actions, errors, warnings, and other events during execution. Each log statement can hold information about the event such as an ID, timestamp, file name, etc. [1]. Log statements can then be analyzed to find patterns of logs that describe specific processes, and then those patterns can be used for anomaly detection, performance testing, security issues, and more [9]. However, log statements can originate from many different places within a distributed system (a large-scale complex set of uncoupled and unrelated nodes that interact asynchronously across the network [14]) is where multi-source log analysis becomes important. With the complexity and scale of a distributed system, many different log types are generated, such as system, event, security, and user logs. It can be difficult to trace issues and analyze problems manually.

Our research's motivation is the absence of a general method for multi-file log matching in a distributed system, as stated by Landauer et al. in the survey [9]. Log clustering has traditionally only been applied to the scope of a single file, thus making our research novel and necessary [9]. Multi-file log analysis is relatively new, few papers discuss it, and many of these papers focus solely on multi-file log analysis algorithms to solve problems related to cybersecurity [21]. However, logs can be utilized to solve issues outside of security like finding resource bottlenecks [8], monitoring system performance [7,24], finding event

sequences [5], diagnosing failures [3], and checking system upgrade results [8]. The growing number of distributed systems and log types over the past years necessitates a log clustering algorithm that would work across multiple log files for arbitrary log types [9].

Before any clustering can be done, the log statements must be converted into a structured form and potentially compressed. Some of the most popular approaches include log parsing [27], log compression [11], and natural language processing [2]. One study covers 13 log parsing techniques applied to 16 different data sources [27]. Since the data sources are quite large, a technique to compress the logs into a form that is easier to work with is also necessary [11]. The log parsing technique (Drain) chosen by this study uses heuristics to separate log messages into groups by analyzing constant and variable tokens. This technique is also applied to distributed systems, so the applicability of a heuristics approach on a multi-source log file system is supported. On the other hand, a natural language processing approach could be better suited to parse many types of logs, but since this technique has not been applied to a distributed system, there is no evidence that this would be more effective.

Log clustering survey [9] describes different methods of static and dynamic clustering and the ways in which they are applied in cybersecurity. This survey results give the best algorithms and techniques for outlier detection, anomaly detection, and log parsing. Together with Landauer et al. article on dynamic log file analysis [10], it shows how the combination of static and dynamic clustering proves effective. However, both tend to be exclusively applicable to cybersecurity and anomaly detection by analyzing outliers and sequences of logs. Our generic approach for log clustering applies beyond cybersecurity; it can help to build clusters between files, which is not a focus in either of these works.

Other log clustering approaches use signature extraction, statistics, and pattern recognition to cluster and perform analysis on logs [16,1,4,25,12]. Some of these methods generate templates for logs and use those templates to find log statement patterns [4,25], while others group logs together with the same template [16,1]. However, a better algorithm would take into account the variable pieces of logs like any ID's, event codes, and file names. An approach that takes this into account finds the frequent log constants, maps those constants to a specific number, then generates structured sessions based on execution order [13]. This approach targets security with the clusters based on execution order, still it provides a very useful method for grouping logs using a key value pairs.

Although the research on clustering log statements is extensive, few studies show interest in multi-source log clustering. The few studies on multi-source logs or logs from distributed systems are most often used to solve problems with anomaly detection, malicious requests, password guessing, brute-force cracking, and attack patterns [22,21,6,15]. Since these studies' focus is mainly cybersecurity, the case-study log files primarily consist of security-formatted logs, rather than a variety of log types.

3 Proposed Method

In order to achieve multi-source logs clustering, there must be some connecting metric between groups of log statements from one file to the other. Finding this connector is the most challenging part, as every set of log data differs from on another in a variety of ways [27]. To begin, we need to find a variable (or a combination of variables) that uniquely identifies a set of one or more statements in a log file. It is possible that variables are not present across multiple files; however, we permit occurrences of the same variable in the two distinct files. Once this variable is identified, we can connect log statements from multiple files, identify their order, and perform analysis.

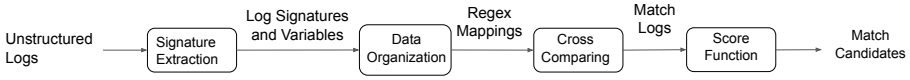

Fig. 1. Overview of our proposed method.

To provide an example, an IP address of a customer will be unique from all other customers, and if you find the same IP address in separate log files within the same distributed environment, you can say with some certainty, there is a correlation between these statements. Our goal then is to search for this IP address in all the log statements across the log files and compare them to each other until all matches are found. We then perform a statistical analysis to determine which match is the best. We cannot simply use the time to cluster, as being from different sources, we cannot know the discrepancies in time zones and communication delays. Including time in our algorithm would also cause unrelated events occurring synchronously would be clustered.

Firstly, we extract the log signature of the log messages to find the candidate variable. Extracting the log signature involves finding the static elements of a given log statement, which are the elements of log output that remain constant every time the line of code that prints the log output is executed [9]. This performs two things for us. First, it gives us a simple clustering of the log messages, which is generally the primary usage of log signature extraction [16]. With that, we have a template of the different messages found in a log file. The second thing finding the log signature does for us is to give us all the variable elements of a log statement. If we know what the static elements are, the reverse yields variable elements [9]. With both the log template and variable components of each log statement, we can proceed to the next step of the method.

Fig. 2. Example signature extraction process to extract log statement variables.

We do not wish to compare every variable in one file with every variable of another, so we need to find some way to narrow down the possibilities. The

straight forward approach is only to consider variables with the same length, however, this has some issues. For example, an IP address can be represented with different lengths, but should still be considered the same variable type. Our proposed method of doing this is to consider the regular expression of the variables. A regular expression is a string used to describe a text pattern in another string (ie. `[a-z]+` describes `"hello"`), and would not describe `"12345"` [23]. These regular expressions need not be defined the same as traditional regular expressions are, but capture enough information to distinguish variables from each other. The generation of these regular expressions is beyond the paper's scope, so the length can be used as an alternative if needed. If two variables are to have matches, they must have the same regular expression. With this being the case, we do not have to check between variables that do not have the same regular expression. The data structure we used is a nested map to store the information for a single file's variables. We mapped the log templates (the key of which could simply be the static component of the message) to each of the variables in that template, which are then mapped to each of the possible regular expressions that that variable can take on. We then map the regular expressions to the actual values found in the data, along with what line(s) the values were found on.

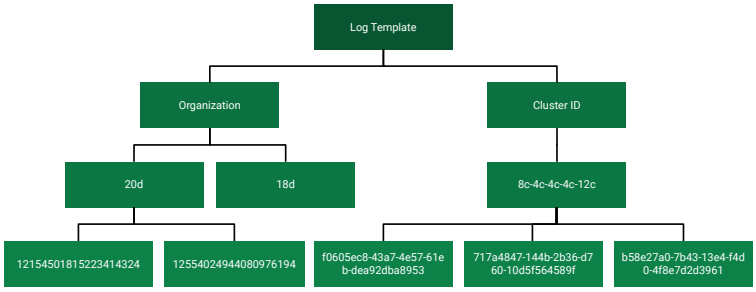

Fig. 3. Tree representation of our mapping structure.

With this, we can now begin matching. We wish to create matches between variables that share the same regex. Since some variables may have more than one regex, only one regex match is required to become a candidate match. When we find two variables with the same regex, we create a match object, which also stores some relevant information regarding the two variables. We compare every instance of a variable to every value-instance of the other log file and create an entry for each variable regardless if there was a match or not. For each value, we store the total number of occurrences in each file. For example, if it was in each file once, then we would store 2. We also store the number of matches. A match is the number of occurrences of the value in one file multiplied by the number of occurrences in the other. This is because we can not know what the true match is without more information for each duplicate value in one file, so we record all possibilities. A variable will likely appear in multiple different log templates across a file, so there will likely be duplicate match groups as well. These can be

avoided by either performing intra-file matches to identify repeating variables or considering the variable's key name, if available. Provided is an example in pseudo-code describing the process of matching the log templates to each other.

```
for each log template a and b in log files A and B:
    if regex(a) == regex(b):
        create match
        for every value in a, b:
            match.value_count += 1
            if a == b:
                match.match_count += 1
```

Listing 1.1. Primary matching code example

With all of the matches throughout the files found, we can narrow down the candidate list. For each item in the match list, we can find the number of unique values by counting the number of distinct values in the match list. The number of unique matches can be calculated by counting the number of values in the match list with a match count greater than zero. From there, we can find the ratio of unique matches to total matches and unique values to total values. Variable ratios help give us an indication of how well the match is. Those with a low ratio have many repeated values and indicate no real correlation between two log statements with the same variable value. Those with high ratios indicate that they are more likely to be a strong indicator of a connection between statements. We also compare the number of unique matches with the number of unique values. Having a high ratio of unique to total matches is not enough if there is only one match over a thousand values, so we ensure that there is not an insignificant amount of matches. Along with this point, we also want to ensure that there is not an insignificant number of values. A match with 10 total values across tens of thousands of lines of logs does not cut it, so a ratio of total values to total number of lines is taken. All these variable ratios are then weighted to produce a score, and the variable match with the highest score is selected as the primary candidate to cluster between files. Definition 1 defines the score function in a more concise manner. The values of the weights depend on the data set given, however, from various tests we conducted, we have found that simply leaving the weights at 1.0 provided adequate results.

Definition 1 (Score function).

$$Score(A, G) = (w_m + (A_{um}/A_{tm})) + (w_v \times (A_{uv}/A_{tv})) + (w_g \times (A_{tv}/G))$$

Where A is the match object, and each of it's subvalues, A_{um},A_{tm},A_{uv}, and A_{tv} represent the unique matches, total matches, unique values, and total values respectively. G is the global value count. w_m, w_v, w_g represent the weights for each of the ratios.

4 Case Study

We performed a blind case study on RedHat's logs from their distributed production system [17] [18]. The system is based on multiple heterogeneous machines together with Kafka. We selected two machines for our analysis called *ccx-data-pipeline* [20] and *ccx-data-aggregator* [19]. We include a figure of the whole data

pipeline from which the files originate. The pipeline file contained 133,000 log lines and was stored in plain text, and we identified 4 dynamic variables: Time, ClusterID, Organization, and partition. The aggregator file contained 196,000 lines of logs, was stored in JSON format, and we identified 4 dynamic variables: time, cluster_id organization, and partition. In the pipeline, file were logs of sent messages through the Kafka cloud system. These messages would have two primary attributes, organization and cluster ID. Both organization and cluster ID combined would contribute to a unique identifier for a given message, but both on their own had the possibility to be repeated. The organization had numerous repeating values, whereas cluster ID had very few repeating across different organizations. The other file, aggregator, recorded log messages of the messages being received. The attributes organization and cluster were also recorded here.

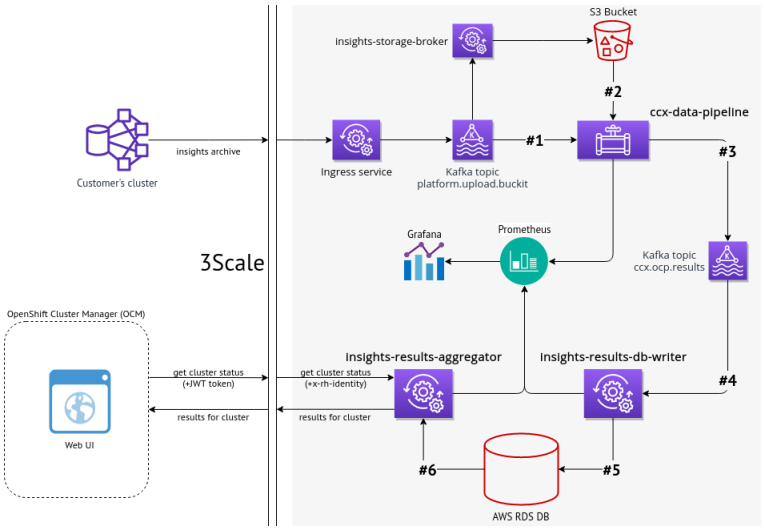

Fig. 4. RedHat Data Pipeline Flow [18]

We then created tests for small portions of each log file at each phase of the method. After this, we performed a full test on entire log files with line counts in the hundreds of thousands. Our tests' results were that the match between the cluster ID variables in aggregator and pipeline was the primary cluster candidate. This is what the answer should have been, as cluster ID is the most unique variable of each message.

Provided are the specific results of one of our tests. Cluster ID scored the highest, greater than the next highest candidate, organization, by a factor of 10^6. Another variable we previously did not account for found via our algorithm was partition. Partition was a 1-2 digit variable which appeared semi-frequently throughout both files, but was only specific to the machine which created the log file, and had no correlation between the two files. Since the variable was only a few digits long and was repeated regularly over the course of hundreds of thousands of lines, it received a very low score.

Table 1 provides our results. The name of the match candidates is each variable's names from the pipeline and aggregator files, respectively. The score represents how close to a perfect match the candidate is. As can be seen, the cluster match performed with a very high score, while the others had very low scores, indicating that the cluster match is the best for clustering.

Match Candidate	Score
ClusterID, cluster_id	0.108
Organization, organization	9.834×10^{-6}
partition, partition	1.131×10^{-15}

Table 1. Case study match candidate scores

Threats to Validity: We expected the score for both cluster ID and organization to be as higher than the actual reported value, but after examination of the log files, it became clear why. The aggregator file would print multiple messages for each received message, with each including the organization and cluster ID. This meant that the aggregator would print organization and pipeline at least 5 times for every message sent from the pipeline. These repeated values skewed the ratios for unique values and matches. This ultimately posed no problem to our data set; however, another data set may be afflicted with false-positives from such a situation. Our proposed method always provides an output, so whether there is a real connection between two files cannot be determined, only if there are any matches, which match appears to be the most relevant. However, if the weighted score for the primary output is exceptionally low, then there's a chance there is no true correlation. Of course, to what extent the score must be to be considered exceptionally low depends on the data set. While intended to work on generic data, some knowledge of the data set would be useful to accurately understand the results.

5 Conclusion

This paper proposed a unique method for multi-file log matching in a distributed system. It provides researchers with a general method for further analysis in distributed systems using data mining techniques. This method enables to analyze security leaks, patterns, and intrusions across the distributed system using a centralized approach. We demonstrated the approach on a case study involving two production systems and proved that our method produces correct results on a generic system. Our long term goal is to provide methods for message tracing and automated security checks in the distributed system.

Our test cases set the weights ratio to `1.0`; however, this requires deeper knowledge about the data set to set weights. Machine learning would be optimal for training the weights prediction for a particular data set.

The method we propose attempts to cluster log statements together with a single variable. In future work, we will cluster log statements based on multiple variables. This would prove useful for data sets where events have more than a single identifier. In our data set, each pair of organization and cluster ID were unique, whereas when only considering the individual values, organization and

cluster ID was not unique. Even after clustering based on cluster ID, there may be a few incorrect cluster points. A combination of both organization and cluster ID would be the most accurate. If cluster ID were not unique, the single variable clustering algorithm would not be sufficient. Our approach could also cluster more files, and performance needs to be assessed.

Acknowledgments
This material is based upon work supported by the National Science Foundation under Grant No. 1854049 and a grant from Red Hat Research.

References

1. Aharon, M., Barash, G., Cohen, I., Mordechai, E.: One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs. In: Machine Learning and Knowledge Discovery in Databases. pp. 227–243. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
2. Amato, Flora, C., Giovanni, M., Antonino, M., Francesco: Detect and correlate information system events through verbose logging messages analysis (2019). https://doi.org/10.1007/s00607-018-0662-1
3. Chuah, E., Kuo, S., Hiew, P., Tjhi, W., Lee, G., Hammond, J., Michalewicz, M.T., Hung, T., Browne, J.C.: Diagnosing the root-causes of failures from cluster log files. In: International Conference on High Performance Computing. pp. 1–10 (2010)
4. Debnath, B., Solaimani, M., Gulzar, M.A.G., Arora, N., Lumezanu, C., Xu, J., Zong, B., Zhang, H., Jiang, G., Khan, L.: Loglens: A real-time log analysis system. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). pp. 1052–1062 (2018)
5. He, P., Zhu, J., Zheng, Z., Lyu, M.R.: Drain: An online log parsing approach with fixed depth tree. In: 2017 IEEE International Conference on Web Services (ICWS). pp. 33–40 (2017)
6. Jia, Z., Shen, C., Yi, X., Chen, Y., Yu, T., Guan, X.: Big-data analysis of multi-source logs for anomaly detection on network-based system. In: 2017 13th IEEE Conference on Automation Science and Engineering (CASE). pp. 1136–1141 (2017)
7. Juvonen, A., Sipola, T., Hämäläinen, T.: Online anomaly detection using dimensionality reduction techniques for http log analysis. Computer Networks **91**, 46 – 56 (2015). https://doi.org/https://doi.org/10.1016/j.comnet.2015.07.019
8. Kubacki, M., Sosnowski, J.: Exploring operational profiles and anomalies in computer performance logs. Microprocessors and Microsystems **69**, 1 – 15 (2019). https://doi.org/https://doi.org/10.1016/j.micpro.2019.05.007
9. Landauer, M., Skopik, F., Wurzenberger, M., Rauber, A.: System log clustering approaches for cyber security applications: A survey. Computers & Security **92**, 101739 (2020). https://doi.org/https://doi.org/10.1016/j.cose.2020.101739
10. Landauer, M., Wurzenberger, M., Skopik, F., Settanni, G., Filzmoser, P.: Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection. Computers & Security **79**, 94 – 116 (2018). https://doi.org/https://doi.org/10.1016/j.cose.2018.08.009
11. Liu, J., Zhu, J., He, S., He, P., Zheng, Z., Lyu, M.R.: Logzip: Extracting hidden structures via iterative clustering for log compression. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering. p. 863–873. ASE '19, IEEE Press (2019). https://doi.org/10.1109/ASE.2019.00085

12. Lu, J., Liu, C., Li, F., Li, L., Feng, X., Xue, J.: Cloudraid: Detecting distributed concurrency bugs via log-mining and enhancement. IEEE Transactions on Software Engineering pp. 1–1 (2020)
13. Lu, S., Wei, X., Li, Y., Wang, L.: Detecting anomaly in big data system logs using convolutional neural network. In: 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress. pp. 151–158 (2018)
14. Nagaraj, K., Killian, C., Neville, J.: Structured comparative analysis of systems logs to diagnose performance problems (2011)
15. Pei, K., Gu, Z., Saltaformaggio, B., Ma, S., Wang, F., Zhang, Z., Si, L., Zhang, X., Xu, D.: Hercule: Attack story reconstruction via community discovery on correlated log graph. In: Proceedings of the 32nd Annual Conference on Computer Security Applications. p. 583–595. ACSAC '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2991079.2991122
16. Pokharel, P., Pokhrel, R., Joshi, B.: A hybrid approach for log signature generation. Applied Computing and Informatics (2019). https://doi.org/https://doi.org/10.1016/j.aci.2019.05.002
17. RedHat: Logscraper. https://cloudhubs.github.io/logscraper/ (2020), online; accessed 30 July 2020
18. RedHat: Logscraper pipeline. https://cloudhubs.github.io/logscraper/Pipeline.html (2020), online; accessed 30 July 2020
19. RedHat: Redhatinsights aggregator. https://github.com/RedHatInsights/insights-results-aggregator (2020), online; accessed 30 July 2020
20. RedHat: Redhatinsights pipeline. https://github.com/RedHatInsights/ccx-data-pipeline-monitor (2020), online; accessed 30 July 2020
21. Shu, X., Smiy, J., Daphne Yao, D., Lin, H.: Massive distributed and parallel log analysis for organizational security. In: 2013 IEEE Globecom Workshops (GC Wkshps). pp. 194–199 (2013)
22. Sun, Y., Guo, S., Chen, Z.: Intelligent log analysis system for massive and multi-source security logs: Mmslas design and implementation plan. In: 15th International Conference on Mobile Ad-Hoc and Sensor Networks. pp. 416–421 (2019)
23. Tania, K.D., Tama, B.A.: Implementation of regular expression (regex) on knowledge management system. In: 2017 International Conference on Data and Software Engineering (ICoDSE). pp. 1–5 (2017)
24. Wurzenberger, M., Skopik, F., Landauer, M., Greitbauer, P., Fiedler, R., Kastner, W.: Incremental clustering for semi-supervised anomaly detection applied on log data. In: Proceedings of the 12th International Conference on Availability, Reliability and Security. ARES '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3098954.3098973
25. Xu, W.: System Problem Detection by Mining Console Logs. Ph.D. thesis, USA (2010)
26. Zeng, Q., Duan, H., Liu, C.: Top-down process mining from multi-source running logs based on refinement of petri nets. IEEE Access **8**, 61355–61369 (2020)
27. Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., Lyu, M.R.: Tools and benchmarks for automated log parsing. In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice. p. 121–130. ICSE-SEIP '19, IEEE Press (2019). https://doi.org/10.1109/ICSE-SEIP.2019.00021