Supplementary Material for: A Survey on Automated Log Analysis for Reliability Engineering

SHILIN HE, Microsoft Research
PINJIA HE, Department of Computer Science, ETH Zurich
ZHUANGBIN CHEN, TIANYI YANG, YUXIN SU, and MICHAEL R. LYU,
Department of Computer Science and Engineering, The Chinese University of Hong Kong

A BEST CURRENT PRACTICES

Log plays an essential role in industrial companies for its carried information, e.g., log management tools have been integrated in various cloud platforms. Although this survey is mainly research-oriented, in this section, we intend to discuss the current industrial best practices based on our experience and surveyed papers and articles. Note that the practices do not indicate the underlying technologies used by specific companies and we do not aim to include all the best practices. Most existing practices concentrate on the logging, log collection, and monitoring aspects, which are summarized into six best practices as follows.

Practice 1: Always follow the logging standards. It is crucial to follow the standards of logging during development, otherwise the produced logs would be hard to maintain, search, and analyze. For example, the following logging standards are shared by various systems: (1) Timestamp: timestamp helps developers understand the sequential relationship among log events. Using correct timestamps (UTC/timezone adjusted) is necessary for debugging and analytic purposes. (2) Verbosity levels: proper verbosity levels ease log parsing and searching. In addition, aggregating logs by verbosity level is beneficial. (3) Format: log format is highly correlated with the parse and search procedures while most people might ignore. It is recommended to structure logs following an agreed standard (e.g., in JSON format) within the same project group. (4) Log message: meaningful log messages facilitates the identification of the correct root cause for a failure. To construct meaning log messages, it is suggested to avoid duplicate logging descriptions (e.g., by assignment a unique ID to each logging statement).

Practice 2: Keep proper quantity of log messages. Controlling the number of logging statements in the source code is very tricky. If logging too little, then we may not have adequate information for problem diagnosis. On the contrary, if logging too much, then we can easily get overwhelmed by the huge volume of logs and problem diagnosis is like looking for needles in haystacks. Moreover, too many logging statements could lead to unnecessary performance overhead [5, 17]. Hence, it is crucial to keep proper quantity of log messages.

Practice 3: Trace log cycle across services. The widely employed micro-services and components in industry bring great challenges to the tracing of a request life cycle. Engineers may get overwhelmed by interleaving logs generated by different requests, hindering problem diagnosis in practice. Therefore, it is necessary to record the event ID information during log collection. A recent

0360-0300/2021/07-ART130 $15.00
https://doi.org/10.1145/3460345

study [11] revealed that many difficulties in automated log analysis are caused by logs of low quality, and a typical example is the missing of event IDs. Without event IDs, we cannot link log messages related to the operations of a single request together. Event ID is also widely leveraged in recent research work [9, 14, 16]. Typically, developers add a unique event ID to the head of each log message, and use it during the inter-service communications.

Practice 4: Aggregate logs to a centralized location. Since logs are often generated in different services and components separately, it is important to aggregate the logs to a centralized location for convenient log search and analysis. Moreover, there are many practical challenges to consider after aggregating logs, including log file name configuration, retention policy, storage size, backup strategy, and so on. We take retention policy as an example. The increasing volume of log data boost the storage cost and query time, thereby an appropriate retention policy is highly in demand. In addition, logs of different types might have different retention requirements. For example, application logs for troubleshooting can be kept for only a few weeks, while audit logs or transaction logs require much longer retention times. Therefore, the retention policy should be flexible.

Practice 5: Safeguard the collected logs. Due to the abundant information (e.g., database address) stored in logs, log data is often under a high risk of attacks. Adversaries tend to remove their trails of action, and an intuitive way is to remove any logs that might reveal their actions. Therefore, safeguarding the log files in case of an attack is a must. Because of similar reasons, industrial companies are usually not willing to share their log data to the public. Apart from setting up proper safeguarding strategies for the log storage, it is a highly desirable practice to avoid logging any confidential information (e.g., password). Recording suspicious human activities (e.g., failed authentication) and system behaviors (e.g., spikes in resource consumption) are also crucial to log safeguarding.

Practice 6: Integrate logs with monitoring metrics. In a large log dataset, manually inspecting every log message brings heavy workload to the developers. The log information is useful only when an issue is identified. In practice, logs are often used together with other signals such as monitors. A monitor aggregates information from multiple sources and often serves as the first step towards troubleshooting. The monitoring metric could be a status count extracted from raw logs (e.g., the number of error events), a health indicator of service performance (e.g., the number of successful requests) or a resource monitor (e.g., CPU and memory consumption). Generally, monitors are embedded in a monitoring dashboard for real-time monitoring and alerting. Because of these monitors, troubleshooting by searching and analyzing logs becomes more effective.

B FUTURE DIRECTIONS

In this section, we discuss promising future directions. Particularly, we start with concluding future directions for each topic reviewed in this survey: logging, log compression, log parsing, and log mining. Then we share our thoughts on the next-generation log analysis framework, which might shed light on future studies. In particular, we believe the next-generation log analysis should be "autopilot" via better human-computer interaction and an end-to-end analytic structure. Note that these directions are mainly recommended by the authors or reviewers of this article, and we believe there are much more interesting topics beyond what we have discussed in the following.

Logging

Although logging has been widely studied in recent years, we believe there is still much room for further improvement. In the following, we introduce two future directions: analysis-oriented logging and logging convention enforcement.

Analysis-oriented Logging. As the foundation of automated log analysis, current logging practices mostly focus on characterization and recommendation of logging statements (i.e., where, what, and how to log). Meanwhile, a recent industrial study [1] pointed out that the inconsistent presentation format with-in and among teams poses a significant challenge in developing automated log analysis tools. In addition, even an effective log mining algorithm could generate meaningless results if the software runtime information is not well-documented in the collected logs. Thus, a crucial direction is designing advanced logging mechanisms that can coordinate with the subsequent log analysis steps during the development.

Logging Convention Enforcement. Although there are no golden rules of logging, developers typically intend to have consistent logging practice in the same project. For example, senior developers in a project may define a set of rules with examples and ask other developers to follow. However, it is difficult for junior developers to get familiar with all the suggested rules, which often change when they are involved in new projects. Thus, it is promising to design automated tools (e.g., lightweight static analysis tools) to recommend logging practice and report potential violations of the "missing"/"incorrect" logging code.

Log Compression

Query-efficient Log Compression. Different logs may play different roles and have different storage requirements. For example, failure logs could be frequently queried by developers and the corresponding log compressor should promptly respond to the query request. Therefore, how to incorporate the query demands into log compression becomes vital to log service providers. Particularly, several potential sub-problems could be: how to predict the user query request, how to store logs in a systematically modularized way for different demands, etc.

Log Parsing

Distributed Online Log Parsing. Existing approaches [7, 8] provide offline parallel mechanisms to accelerate the log parsing process. However, there is a lack of distributed online parsing solution in large-scale distributed systems. The desired log parser should have multiple running threads in both the master and the working nodes and parse the log messages in a streaming manner. A trade-off will be considered between parsing accuracy and efficiency via an event template synchronization mechanism. With a distributed online parser, we could generate structured log messages on the fly, which facilities both log compression and log mining tasks.

Log Mining

Although being extensively studied, log mining is still very challenging due to complex problem definition and poor data quality in different scenarios. In the following, we summarize insights inspired by the surveyed papers and propose some promising future directions. Unlike some other research topics (e.g., log parsing) that are relatively mature, log mining still leaves a lot of space to explore.

Concept Drift-adapted Log Mining. Most of the existing approaches train models with historical log data offline. However, modern software systems continuously undergo feature upgrades and system renewal; hence, the patterns of logs may drift accordingly [18]. Common strategies of online updating, e.g., periodically retraining the model, suffer from high false positive rates, and strategies like zero-positive learning [6] have limited generalization ability. Online learning and incremental learning that fit the log mining scenario are appealing research directions.

Interpretable Log Mining. Whether a log analysis algorithm can provide interpretable results is vital for administrator and analysts to trust and act on the automated analysis. Some traditional machine learning algorithms possess the merit of interpretability [10, 12, 13, 15]. However, deep learning models, though having achieved impressive performance, are criticized as black-box oracles. Therefore, the community of log analysis should continue pursuing more explainable and trustworthy algorithms, for example, by utilizing attention mechanisms [2].

Log Completeness. Log completeness [4] measures whether developers have seen enough logs for different log mining tasks. Log completeness is important because (1) it facilitates measuring the reliability of the log mining results, and (2) it is strongly related to the heuristics that may address the scalability issues of log mining [3]. However, little work has been done on log completeness, which makes it a promising and important future direction.

Next-generation Log Analysis Framework

Log analysis, as a core component in Artificial Intelligence for IT Operations, is crucial to the reliability engineering of software systems. Particularly, the demand for automated log analysis draws dramatic attention as the increasing needs of cloud computing. We believe that the next-generation log analysis would be the "autopilot" in cloud environment. We list as below two major directions that the next-generation log analysis framework may contain.

Human-Computer Interactive Log Analysis. Current automated log analysis research facilitates the failure diagnosis process by providing insightful decisions and clues for root cause analysis. In essence, developers are still at the core of failure diagnosis. For example, the diagnosis results should be confirmed and then debugged by developers. However, there is a lack of interactions between the advanced analysis techniques and developers-in-charge. Besides, the diagnosis process and results are seldom reused in the future log analysis. Therefore, it is interesting and promising to involve developers into the loop of automated log analysis. It will be promising to build an interactive dialogue system with automated analysis tools. The automated log analysis tools first provide suggestions to the developers for decision making. Then the developers can request for additional information of failures, and the diagnosis process with human involvement can be recorded and reused to improve the automated log analysis tools.

Intelligent and End-to-end Log Analysis. Current automated log analysis contains several consecutive steps, but they are not working seamlessly together. In particular, they do not form an end-to-end framework, indicating the possible loss of some critical information in the middle. In our vision, the ultimate goal of the next generation log analysis is an intelligent and end-to-end framework. In this framework, logging statements are automatically suggested by considering the failure diagnosis demand and the potential machine learning models used in log mining. Then, the framework intelligently stores raw logs following practical requirements. Logs are processed without information loss and are correlated with human-observable failures for diagnosis. Moreover, the framework would automatically generate failure reports by aggregating essential information from all analyzed system logs. It should also auto-repair the detected failures by observing experiences from human experts.

REFERENCES

[1] Titus Barik, Robert DeLine, Steven Drucker, and Danyel Fisher. 2016. The bones of the system: A case study of logging and telemetry at microsoft. In *Proceedings of the IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C'16)*. IEEE, 92–101.

[2] Andy Brown, Aaron Tuor, Brian Hutchinson, and Nicole Nichols. 2018. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In *Proceedings of the First Workshop on Machine Learning for Computing Systems*. 1–8.

[3] Nimrod Busany and Shahar Maoz. 2016. Behavioral log analysis with statistical guarantees. In *Proceedings of the IEEE/ACM 38th International Conference on Software Engineering (ICSE'16)*. 877–887.

[4] Hila Cohen and Shahar Maoz. 2015. Have we seen enough traces? In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*. 93–103.

[5] Rui Ding, Hucheng Zhou, Jian-Guang Lou, Hongyu Zhang, Qingwei Lin, Qiang Fu, Dongmei Zhang, and Tao Xie. 2015. Log2: A cost-aware logging mechanism for performance diagnosis. In *Proceedings of the 2015 USENIX Annual Technical Conference (ATC'15)*. 139–150.

[6] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. 2019. Lifelong anomaly detection through unlearning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*. 1283–1297.

[7] Min Du and Feifei Li. 2018. Spell: Online streaming parsing of large unstructured system logs. *IEEE Trans. Knowl. Data Eng.* 31, 11 (2018), 2213–2227.

[8] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. 2017. Towards automated log parsing for large-scale log data analysis. *IEEE Trans. Depend. Secure Comput.* 15, 6 (2017), 931–944.

[9] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'18)*. 60–70.

[10] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining invariants from console logs for system problem detection. In *Proceedings of the USENIX Annual Technical Conference (ATC'10)*. 1–14.

[11] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2017. Experience report on applying software analytics in incident management of online service. *Autom. Softw. Eng.* 24, 4 (2017), 905–941.

[12] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B. Dasgupta, and Subhrajit Bhattacharya. 2016. Anomaly detection using program control flow graph mining from execution logs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 215–224.

[13] Weiyi Shang, Zhen Ming Jiang, Hadi Hemmati, Brain Adams, Ahmed E. Hassan, and Patrick Martin. 2013. Assisting developers of big data analytics applications when deploying on hadoop clouds. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 402–411.

[14] Risto Vaarandi and Mauno Pihelgas. 2015. LogCluster—A data clustering and pattern mining algorithm for event logs. In *Proceedings of the 11th International Conference on Network and Service Management (CNSM'15)*. 1–7.

[15] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Online system problem detection by mining patterns of console logs. In *Proceedings of the 9th IEEE International Conference on Data Mining (ICDM'09)*. 588–597.

[16] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP'09)*.

[17] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael Mihn-Jong Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. 2012. Be conservative: Enhancing failure diagnosis with proactive logging. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI'12)*. 293–306.

[18] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'19)*.