Performance Diagnosis with Logs and Metrics

Jae Jimmy Wong
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
jwong23@ncsu.edu

ABSTRACT

This paper introduced two approaches on analyzing performance bugs for system logs and metrics: keyword-based and two-indicator. Keyword-based approach ranks logs based on the weighting of automatically selected keywords. Two-indicator approach utilizes the hotness of timing and the uniqueness of wording of logs to rank them.

CCS CONCEPTS

• Software and its engineering → Software testing and debugging.

KEYWORDS

Debugging, Performance, Logging

ACM Reference format:

Jae Jimmy Wong. 2023. Performance Diagnosis with Logs and Metrics.

1 Introduction

Performance bugs are hard to identify without extensive experience of solving similar situation, because they are usually hard to be reproduced on testing environment and do not produce clear indicator for its source. Developers often need to figure out the origin of a performance bug by studying system logs and performance metric records, however these logs and records are filled with noises that are either unrelated to the bug or unhelpful for the debugging. This problem is much more serious in a distributed system since a bug may incur performance reduction on other components of the system. The effect propagation can lead people to misinterpret the source of bug and try on wrong fixing methods. The result is that debugging performance problems in distributed systems cost lots of developer's time. There are also many performance issues that is discovered after a long time, which causes significant wasting of computer resources.

This paper proposed two approaches that automatically analyze the logs and metrics, then produce a ranking of each line of log. Software developers can utilize this ranking to find out performance-related with less effort compared to manual review all the log files.

2 System Design

To automatically analyze the essential information of log files from distributed systems, I first experimented with a keyword-based approach. The detail of this approach is described in section 2.1, which includes its shortcoming. Aiming for a better result, the second approach with a focus on resource hotness and word uniqueness has been created. These attributes with be explained in section 2.2, as well as the design choices of the second approach.

2.1 Keyword-based Approach

Most system logs include human-readable descriptions, and these descriptions inherit some characteristic of nature language. One of which is that it utilizes many preposition words. These preposition words provide less information compared to other words. Thus, the first approach tries to eliminate these words before determining the importance of one line of log.

Specifically, the analyzing algorithm first selects a fixed number of keywords from each line of log. Then, the algorithm reads the system metric for each second. If the metric increased 1 compared to last second, all keyword appeared in the log of that second gains 1 additional weight. By repeating this process for all the recorded metrics, a final weight of each keyword is acquired. The sum of the final weight of the keywords of a line of logs is the ranking score of that log. Log with higher ranking score is more related to system metric variation, which is taken as a sign of performance bug.

This approach works well on the situation that each line of log has similar length. However, it comes short when some of the lines is short so that the fixed number of

keywords will force some unimportant words being picked. The result is that short line of log with repetitive words will gain higher weight while it has no relation with performance bugs.

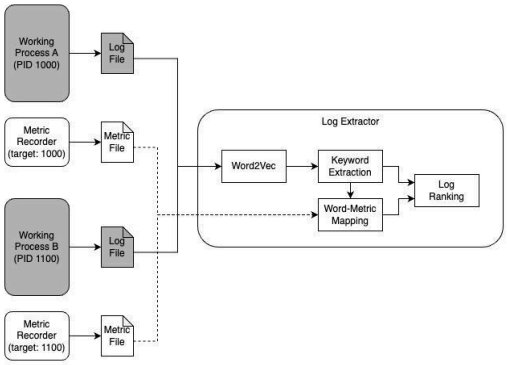

Figure 1: **System Architecture for experimenting the Keyword-based approach.**

2.2 Two-Indicator Approach

The preliminary results of the keyword-based approach shows that it ranks too high on some lines of log. More importantly, these logs can be easily eliminated by a human log reviewer. One of the most important reasons is the similarity of the log. Most of the software logs are repetitively report the status of the system. These logs provided minimum information on what goes wrong inside the system, because they are all the same. Based on this observation, I create an indicator named “uniqueness”. This indicator is currently calculated in a naïve way, which is one divided by the appearance count of the word.

Most performance bug create spikes in the usage of system resource. I go with the assumption that the log that is closer to the time the spike happened is more likely to be related to a bug. In practice, the algorithm first determines “hotspots” in the time span of execution. A hotspot means there are a change of CPU or memory usage that is more than the average of changes plus a fixed number of standard deviation of changes. The probability dense function of normal distribution with the time of a hotspot as its center is used to calculate the “hotness” of a point of time, so logs that near a hotspot in timeline has higher hotness.

The final score of a line is the product of uniqueness and hotness because this produces the result what fits the goal. That is,
log with either low uniqueness or low hotness has low score, and log with both high uniqueness and high hotness has high score.

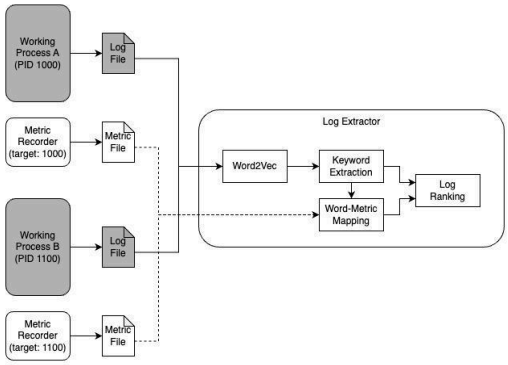

Figure 2: **System Architecture for experimenting the Two-indicator approach.**

3 Evaluation

I picked four distributed performance bugs from previous research and reproduce them with automated scripts. The experiments are conducted on a Google Cloud Platform computing instance with 4 CPU core and 8192 MB memory. I used five as the number of keywords for the keyword-based approach. Results show that the approach ranked the most important log at 365th in a total of 6048 lines of logs and 398th in a total of 780 line of logs. For the two-indicator approach, three standard deviation was the threshold for hotspot, and it has better ranking result. The ranking of the approach has the most importance line of log in the first three places in all four experiments. The implementation is open sourced at http://github.com/jjwong0915/perf-dx.git.

Bug ID	The Rank of Most Importance Log	Total Log Count
HDFS-11252	3	404
HDFS-91106	1	389
HDFS-7005	3	30
HDFS-1490	3	75

Table 1: **The Ranking Result comes from the Two-indicator Approach.**

4 Related Work

Detecting Large-Scale System Problems by Mining Console Logs[1] by Xu et al. and PerSig[2] by He et al. both utilized the technique of transforming free-text console logs into numeric features, which is also used in the

keyword-based approach. However, this paper focused on black-box approaches that are neither depends on source code availability nor based on binary augmentation.

5 Conclusion

The keyword-based log analysis falls short on logs with varying length because of the fixed number of keywords. The two-indicator log analysis utilizes the observation of repetitive logs and resource spikes. It has good result on experiments of reproduced distributed system performance bugs.

REFERENCES

[1] 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 ACM SIGOPS 22nd symposium on Operating systems principles (SOSP '09). Association for Computing Machinery, New York, NY, USA, 117–132. https://doi.org/10.1145/1629575.1629587

[2] Jingzhu He, Yuhang Lin, Xiaohui Gu, Chin-Chia Michael Yeh, and Zhongfang Zhuang. 2022. PerfSig: extracting performance bug signatures via multi-modality causal analysis. In Proceedings of the 44th International Conference on Software Engineering (ICSE '22). Association for Computing Machinery, New York, NY, USA, 1669–1680. https://doi.org/10.1145/3510003.3510110