Tuning Backfired? Not (Always) Your Fault

Understanding and Detecting Configuration-Related Performance Bugs

Haochen He
National University of Defense Technology
Changsha, China
hehaochen13@nudt.edu.cn

ABSTRACT

Performance bugs (PBugs) are often hard to detect due to their non fail-stop symptoms. Existing debugging techniques can only detect PBugs with known patterns (e.g. inefficient loops). The key reason behind this incapability is the lack of a general test oracle. Here, we argue that the configuration tuning can serve as a strong candidate for PBugs detection. First, prior work shows that most performance bugs are related to configurations. Second, the tuning reflects users' expectation of performance changes. If the actual performance behaves differently from the users' intuition, the related code segment is likely to be problematic.

In this paper, we first conduct a comprehensive study on configuration related performance bugs(CPBugs) from 7 representative softwares (i.e., MySQL, MariaDB, MongoDB, RocksDB, PostgreSQL, Apache and Nginx) and collect 135 real-world CPBugs. Next, by further analyzing the symptoms and root causes of the collected bugs, we identify 7 counter-intuitive patterns. Finally, by integrating the counter-intuitive patterns, we build a general test framework for detecting performance bugs.

CCS CONCEPTS

• **Software and its engineering → Software performance**.

KEYWORDS

performance bugs detection, configuration, empirical study

ACM Reference Format:
Haochen He. 2019. Tuning Backfired? Not (Always) Your Fault: Understanding and Detecting Configuration-Related Performance Bugs. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3338906.3342498

1 INTRODUCTION

Performance bugs (PBugs) are notorious for their severe impacts on user experiences and financial losses [6]. However, unlike crash bugs (e.g. stack overflow), performance bugs usually only cause slowdowns, thereby hard to be captured or logged. As a result, it can be difficult to expose PBugs via conventional testing approaches (e.g., unit test).

MySQL #21727
Description: A query of the type: "`SELECT a, b, (SELECT x FROM t2 WHERE y=b ORDER BY z DESC LIMIT 1) c FROM t1`" will get much slower as `sort_buffer_size` is increased. **Related Configuration**: `sort_buffer_size`(Session that needs to perform a sort will be allocated a buffer with this amount of memory.) **Root Cause**: Allocation of memory for the sort buffer at each evaluation of a subquery may take a significant amount of time if the buffer is rather big. In this case, memory accesses are implemented with complicated address manipulation (e.g. system calls and address calculations), and the bug only manifest under large configuration vales (`sort_buffer_size`).

Figure 1: A CPBug in MySQL

Previously, two groups of works have made significant progress in understanding and fixing PBugs [3, 7–10]. The first group focuses on using profiling tools to identify test input with notable time consumption (e.g., GA-Prof [3]). The other group checks suspicious code patterns or memory accesses to detect potential PBugs (e.g., LDoctor [8], LOADSPY [9], Toddler [7], [10]). While effective, both approaches do not offer general detection methodology due to their limited scopes. For instance, as shown in Figure 1, this MySQL bug [1] may not necessarily be sensitive to inputs nor does it exist certain code or memory access patterns.

In this paper, we argue that configuration tuning can be leveraged as a general test oracle for detecting PBugs. First, recent studies [4] suggest that more than half (59%) of performance bugs can be related to configurations. This implies the potentials of using configurations to trigger latent PBugs. Second, when tuning the configuration, users usually have expectations of the possible performance changes. For instance, relaxing the ACID (Atomicity, Consistency, Isolation and Durability) requirement of database should at least maintain the performance, if not better. If, however, the actual performance is against the expectation, it is possible that a PBug exists in the related code segment [2]. Finally, modern softwares are often equipped with rich amount of configurations which indicates a wide coverage of the code segments.

Therefore, to validate practicality of using configuration tuning as PBug test oracle, we first conduct an extensive PBug study on 7 well-maintained softwares including MySQL, MariaDB, MongoDB, RocksDB, PostgreSQL, Apache and Nginx. Through the study, we collect 135 Configuration-related Performance Bugs (CPBugs) from the field. Then, by further analyzing collected CPBugs, we conclude 7 counter-intuitive CPBugs patterns and two findings regarding the

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-5572-8/19/08.
https://doi.org/10.1145/3338906.3342498

root causes and triggering conditions. Based on our observations, we propose our CPBug test framework, CP-Detector.

2 METHODOLOGY

Here, we refer *configuration related performance bugs* (CPBugs) to both performance bugs caused by misconfigurations and ones that can be manifested under specific configurations. We study 7 performance critical and highly-configurable softwares: MySQL, MariaDB, MongoDB, RocksDB, PostgreSQL, Apache httpd and Nginx, as shown in Table 1. These softwares have already been widely used by existing performance bug characteristic studies [4, 5].

Table 1: Applications and bugs used in the study

Category	Application	Studied bugs
Database	MySQL Server	31
	MariaDB Server	32
	MongoDB Server	42
	PostgreSQL	4
	RocksDB	5
Web Server	Apache httpd	21
	Nginx	1
Total		**135**

We first sift through the titles of issues and the software changelogs using heuristic keywords filtering. Then, for each confirmed bug, we further investigate the discussion and bug reports to identify the symptoms, triggering conditions, root causes and possible fixes. Note that we only select bugs that are either confirmed or already fixed by the developers.

By manually analyzing the collected information, first, we want to figure out the relation between CPBugs' symptoms and users' expectation of tuning the configurations, thereby verifying if the *configuration intuitions*(i.e. users' expectations of the performance changes in the configuration tuning process) can be further designed to test oracle. Second, we want to understand why these CPBugs happen and further answer if we can obtain general root cause features to detect CPBugs. Third, we want to find out insights to help us further build up a tool to detect real-world CPBugs.

3 CHARACTERISTIC STUDY RESULTS

We conclude 3 main findings from the results:

Finding 1: Configuration intuitions can be designed to test oracle. As shown in Table 2, we conclude 7 counter-intuitive patterns according to the *configuration intuitions*. The bug case in Figure 1 falls in the *Resource Abuse* because allocating more resource results in worse performance.

Finding 2: Root causes of CPBugs varies. The studied CPBugs is caused by manifold reasons including redundant operations(38%), configuration design issues(19%) , contention issues(18%), and other problems(25%). Existing methods can only detect certain kinds of PBugs (e.g. PBugs caused by redundancies [8]) because of lacking general root causes features. Hence, this finding motivates us to detect general PBugs by exposing them via testing rather than straightly locating them in source codes.

Table 2: 7 counter-intuitive patterns

Patterns	Description	Cnt.	%
Resource Abuse	Allocating more resource results in worse performance.	21	16%
Ineffective Tradeoff	Sacrificing reliability does not result in better performance.	18	13%
Useless Optimization	Turning on optimization makes performance worse.	17	12%
Unexpected Loss	Tuning performance insensitive options degrades performance.	12	8%
More-than-expected Loss	Opening a functional option leads to too bad performance.	41	26%
Unnecessary Waiting	A timeout is always triggered in normal runs.	13	10%
Low Parallelism	Jobs are implemented in single thread in Multi-core CPU.	4	2%
Others	CPBugs that have no relationship with configuration intuitions.	6	3%
Total		135	

Finding 3: Nearly half(44%) of CPBugs can be triggered by benchmark tools. This result is obtained by manually reproducing 43 CPBugs using benchmark tools (e.g. sysbench, ab).

4 DETECTING CPBUGS

Based on the findings above, we propose CP-Detector, an approach to detect configuration related performance bugs via configuration variation. Overview of CP-Detector is shown in Figure 2.

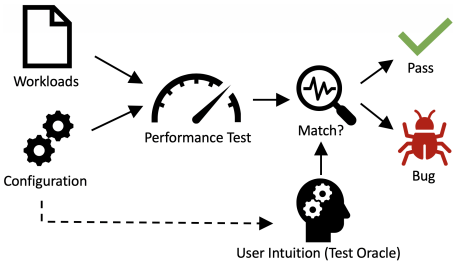

Figure 2: Overview of CP-Detector. *Each test case contains one configuration option and a set of relevant workloads. If the results of running certain workload violate the oracle during the configuration variation, CP-Detector report it as a bug.*

CP-Detector uses benchmark tools to generate workloads and conducts performance tests under different values of target configurations. Previous study [4] has shown 72% of CPBugs are related to only one configurations. Hence, each test case contains only one configuration. Note that the intuitions (e.g. allocating more resources won't hurt performance) of the target configuration should be given in advance (dashed line).

We randomly select 10 CPBugs from 43 reproduced ones, and use CP-Detector to detect them. The result shows that our CP-Detector can successfully detect 4 of them. The remained 6 escape because CP-Detector is not able to generate their necessary workloads. Future work lies on detecting real-world CPBugs.

REFERENCES

[1] Mysql bug 21727. https://bugs.mysql.com/bug.php?id=21727.

[2] Mysql bug 77094. https://bugs.mysql.com/bug.php?id=77094.

[3] Du, S., Qi, L., Poshyvanyk, D., and Grechanik, M. Automating performance bottleneck detection using search-based application profiling. In *International Symposium on Software Testing & Analysis* (2015).

[4] Han, X., and Yu, T. An empirical study on performance bugs for highly configurable software systems. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2016), ESEM '16, ACM, pp. 23:1–23:10.

[5] Jin, G., Song, L., Shi, X., Scherpelz, J., and Lu, S. Understanding and detecting real-world performance bugs. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2012), PLDI '12, ACM, pp. 77–88.

[6] Mayer, M. A 500 ms latency increase could cause 20% traffic loss for google. http://assets.en.oreilly.com/1/event/29/Keynote%20Presentation%202.pdf.

[7] Nistor, A., Song, L., Marinov, D., and Lu, S. Toddler: Detecting performance problems via similar memory-access patterns. In *2013 35th International Conference on Software Engineering (ICSE)* (Los Alamitos, CA, USA, may 2013), IEEE Computer Society, pp. 562–571.

[8] Song, L., and Lu, S. Performance diagnosis for inefficient loops. In *Proceedings of the 39th International Conference on Software Engineering* (Piscataway, NJ, USA, 2017), ICSE '17, IEEE Press, pp. 370–380.

[9] Su, P., Wen, S., Yang, H., Chabbi, M., and Liu, X. Redundant loads: A software inefficiency indicator. In *Proceedings of the 41th International Conference on Software Engineering* (2019), ICSE '19.

[10] Yang, J., Yan, C., Subramaniam, P., Lu, S., and Cheung, A. How not to structure your database-backed web applications: A study of performance bugs in the wild. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (May 2018), pp. 800–810.