Performance Regression Detection in DevOps

Jinfu Chen
fu_chen@encs.concordia.ca
Concordia University, Montreal, Quebec, Canada

ABSTRACT

Performance is an important aspect of software quality. The goals of performance are typically defined by setting upper and lower bounds for response time of a system and physical level measurements such as CPU, memory and I/O. To meet such performance goals, several performance-related activities are needed in development (Dev) and operations (Ops). In fact, large software system failures are often due to performance issues rather than functional bugs. One of the most important performance issues is performance regression. Although performance regressions are not all bugs, they often have a direct impact on users' experience of the system. The process of detection of performance regressions in development and operations is faced with a lot of challenges. First, the detection of performance regression is conducted after the fact, i.e., after the system is built and deployed in the field or dedicated performance testing environments. Large amounts of resources are required to detect, locate, understand and fix performance regressions at such a late stage in the development cycle. Second, even we can detect a performance regression, it is extremely hard to fix it because other changes are applied to the system after the introduction of the regression. These challenges call for further in-depth analyses of the performance regression. In this dissertation, to avoid performance regression slipping into operation, we first perform an exploratory study on the source code changes that introduce performance regressions in order to understand root-causes of performance regression in the source code level. Second, we propose an approach that automatically predicts whether a test would manifest performance regressions in a code commit. To assist practitioners to analyze system performance with operational data, we propose an approach to recovering filed-representative workload that can be used to detect performance regression. We also propose that using execution logs generated by unit tests to predict performance regression in load tests.

CCS CONCEPTS

• **Computer systems organization** → **Software engineering**; *Software performance.*

KEYWORDS

Software performance, performance regression detection, load test

ICSE 42nd, May 23–29, 2020, Seoul, South Korea

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

ACM Reference Format:
Jinfu Chen. 2020. Performance Regression Detection in DevOps. In *ICSE 42nd international conference on software engineering, May 23–29, 2020, Seoul, South Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The rise of large-scale software systems (e.g., Amazon.com and Google Gmail) has posed an impact on people's daily lives from mobile devices users to space station operators. The increasing importance and complexity of such systems make their quality a critical, yet extremely difficult issue to address. Failures in such systems are more often associated with performance issues, rather than with feature bugs [19]. Therefore, performance is an important aspect of software quality.

The goals of performance are typically defined by setting upper and lower bounds for response time of a system and physical level measurements such as CPU, memory and I/O. In order to ensure that the performance of a system meets a requirement, several performance assurance activities are required during development (Dev) and operations (Ops) in the release cycle of large software systems. DevOps is a set of software practices to smooth the deploying release of a software system between development and operations [9]. One of the goals of performance assurance activities in DevOps is to detect performance regressions, i.e., the performance of the same feature in the system is worse than before [2]. Response time degradation and increased CPU usage are typical examples of performance regressions. These performance regressions may directly affect the user experience, increase the resources cost of the system and lead to reputational repercussions. Therefore, detecting and resolving performance regressions is an important task even though the system's performance may meet the requirement. For example, Mozilla has a performance regression policy that requires performance regressions to be reported and resolved as bugs [12].

Due to the importance of performance regression, extensive prior research has proposed automated techniques to detect performance regressions [8, 11, 15, 16]. However, challenges in the practices of DevOps of performance regression detection still exist.

- **No existing performance regression data.** There is no existing data of performance regression introducing code changes.
- **Performance regression detection is too late.** The existing performance regression detection is applied after the system is built and deployed in the field. Large amounts of resources are required to detect, locate, understand and fix performance regressions at such a late stage in the development cycle.
- **Large volume of operational data.** The increasing volume of operational data is beyond the capacity of traditional

human driven practices. It is very time-consuming and error-prone to analyze such large-scale operational logs to understand and detect performance regression.

In this dissertation, we propose several techniques to detect performance regression in DevOps. To collect performance regression data, we propose a statistically rigorous approach to identifying performance regression introducing code changes. In order to detect performance regression as early as possible, we propose an approach to automatically predicting whether a test would manifest performance regressions in a code commit. To assist performance regression detection in operations, we first propose to use recovery workload to detect performance regression based on field execution logs. We also propose an approach to predicting performance regression in load tests using readily available execution logs generated by unit tests.

The rest of this dissertation is organized as follows: Section 2 describes our research hypothesis. Section 3 presents our approaches to the frequent performance assurances during development. Section 4 outlines our proposed approaches to assisting performance assurances with operational data. Section 5 presents the states of the art in practice. Section 6 concludes this dissertation.

2 RESEARCH HYPOTHESIS

The practices of DevOps generate two sources of data about software: development data and filed data. Development data, generated during software development, includes a large amount of historical information of software development. A typical example of development data is source code changes. Field data, generated during operations, consists of available and valuable information about how the system operates. System execution logs and performance metrics are two kinds of typical filed data. In this dissertation, we plan to propose automated approaches to detecting performance regression by mining the large amount of development and field data. Our research hypothesis is follows:

> *Software development historical repositories and field operation logs, which are valuable and readily available resources, can be used to detect performance regression in the context of DevOps.*

We will validate our research hypothesis based on the following aspects:

- **Frequent performance assurances during development:** We start off by examining the root-cause of performance regression introducing code changes. We then propose an approach that automatically predicts whether a test would manifest performance regressions given a code commit.
- **Assisting performance assurances with operational data:** We first study the use of recovery workload to detect performance regression using field execution logs. We then propose an approach to predicting performance regression using readily available execution logs of unit tests.

Therefore, this dissertation intends to 1) detect or predict performance regression introducing source code changes in the early stage during development and 2) assist operators to analyze a huge of filed execution logs to detect system performance regression during operations. In particular, we further divide the scope of our research into two parts of research questions. One is frequent performance assurances during development, another is assisting performance assurances with operational data. For each part, we present the problem that we wish to solve, a brief overview of our proposed approach, the results and research timeline.

3 FREQUENT PERFORMANCE ASSURANCES DURING DEVELOPMENT

Software changes during development, i.e. source code changes, may cause performance regression. If we can detect performance regression as early as possible, the amount of required resources would be significantly reduced if developers were notified whether a code change introduces performance regressions during development. Therefore, to avoid performance regression slipping into operations, we propose to solve two problems.

3.1 Identifying root-causes of performance regressions introducing code changes

Problem: It is difficult to identify the root-cause of performance regression. Prior research has conducted empirical studies on performance bugs. However, performance regressions are not performance bugs. Moreover, there is no existing data of performance regression introducing code changes.

Our proposed approach: We present a statistically rigorous approach [3] to identifying performance regression introducing changes. In general, we extract every commit from the version control repositories (Git) and identify impacted test cases of each commit. Afterwards, we chronologically evaluate the performance of each commit using either the related test cases or performance micro-benchmarks. Finally, we perform statistical analysis on the performance evaluation results to identify performance regression introducing changes. To know the code level root-causes of the performance regression, we follow an iterative approach to identifying the root-causes that the code change introduces performance regression, until we could not find any new reasons.

Result: Our study finds that performance regressions are not rare instances and are often with large effects. A majority of the performance regressions are introduced with bug fixing, rarely with new features. Our manual examination on the performance regression introducing code changes identifies six code level root-causes of performance regressions, where some root-causes (such skippable functions) can be avoided. Developers should always be aware of the possible performance regression from their code change, in order to address avoidable regressions or minimize the impact from un-avoidable regressions.

Expected timeline: This part of work has been published in ICSME 2017 [3].

3.2 Predicting tests that manifest performance regressions at the commit level

Problem: Running all tests to detect performance regressions is an extremely time-consuming task, especially for every commit. Although automated techniques are proposed to detect performance regressions, performance regressions detection remains a task that is conducted after the system is developed and built, as almost the last step in the release cycle.

Our proposed approach: We propose an approach that automatically predicts whether a test would manifest performance regressions given a code commit. We call such tests performance-regression-prone tests. In detail, our approach predicts whether there is regression in a *particular test* of a particular commit (not any test in a commit). To build the prediction model, we first identify performance-regression-prone tests by evaluating all the tests that are impacted by the code changes in prior commits and measuring performance (i.e., response time, CPU and memory usage, I/O read and I/O write) by running each test, repetitively. Afterwards, we build five classifiers, one for each performance metric, modeling whether a test would manifest a performance regression. With such prediction models, after developers commit their code changes, our approach can automatically predict which tests manifest performance regressions for each particular performance metric. Developers can prioritize their performance assurance activities by only running the predicted performance-regression-prone tests to address the performance regressions. The newly executed performance evaluation results are then included in the training data to update the classifiers in order to predict the performance-regression-prone tests in the next new commit.

Preliminary result: Our results show that our approach can predict tests that manifest performance regressions in a commit with high AUC values (on average 0.80). Our approach can drastically reduce the testing time needed to detect performance regressions. In addition, we find that our approach could be used to detect the introduction of six out of nine real-life performance issues from the subject systems during our studied period. Finally, we find that traditional metrics that are associated with size and code change histories are the most important factors in our models.

Expected timeline: We plan to finish this part of work before summer 2020.

4 ASSISTING PERFORMANCE ASSURANCES WITH OPERATIONAL DATA

Software activities generate a large amount of filed execution logs during operations. Such filed logs reflect realistic user behaviors and real workload. Since load test is too complex and time-consuming, i.e., designing a workload, executing a load test, and analyzing the results of a load test. Moreover, load tests cannot represent the field workload due to the frequent workload fluctuation. Therefore, if performance regression is hidden during development, such large valuable operational data can be used to assist performance assurances. Therefore, our goal is to assist performance analyst to generate load tests and predict performance regression in load tests using operational data.

4.1 Recovering filed-representative workload to detect performance regression using field execution logs

Problem: Workload recovered from field execution logs can be used to detect performance regression in practice. However, the recovery of a field-representative workload is a challenging task. First, the process of operations generated a huge amount of execution logs. It is time-consuming and error-prone for performance analyst to analyze such large-scale logs. Second, one must achieve a balance between the level of granularity of the workload and the cost to conduct a load test with such a workload.

Our proposed approach: To recover a field-representative workload being used to detect performance regression, we design approaches to reaching a desirable level of granularity for a workload [4]. In particular, we study the impact of adding different levels of details in the recovered workloads for load tests using field execution logs. We first replicate a prior approach that captures the frequency of basic actions. Afterwards, we design an approach that enriches the basic user actions with their context values. Finally, we design another approach that augments the context with frequently-appearing sequences of actions. our approaches use the frequency of actions, the frequency of enriched actions and the frequency of sequences of enriched actions, respectively, as signatures for each user, then group the users into clusters. Afterwards, we automatically generate load tests based the signature of the representative (center) user in each cluster. We finally study the use of recovery workloads to detect performance regression.

Result: Our results show that our approach that is based on user action sequences with contextual information outperforms the other two approaches and can generate more representative load tests with similar throughput and CPU usage to the original field workload. (i.e., mostly statistically insignificant or with small/trivial effect sizes). Such representative load tests are generated only based on a small number of clusters of users, leading to a low cost of conducting/maintaining such load tests. Finally, we demonstrate that our approaches can detect injected users in the original field workloads with high precision and recall.

Expected timeline: This part of work has been published in ASE 2019 [4].

4.2 Predicting the performance regression in load tests using readily available execution logs of unit tests

Problem: Load testing a system is a required testing procedure. However, first, load test is too complex and time-consuming, i.e., executing a load test and analyzing the results of a load test. Second, load tests cannot represent the field workload due to the frequent workload fluctuation. Finally, replaying field workload is difficult due to the need of dedicated testing environment and tooling support. Therefore, if we can use the field data generated in operations to predict performance regression without need to execute load tests, large amounts of resources can be reduced.

Our proposed approach: We present an approach to mining the relationship between unit tests' logs and load tests' logs, and study the use of readily available execution logs of unit tests to predict performance regression in load tests. In detail, we first extract log events from execution logs generated by unit tests and load tests. We then filter and preprocess the extracted log events to build prediction model to predict performance regression in load tests using readily available logs of unit tests. We expect that performance regression in load tests can be successfully predicted using logs generated by unit tests.

Preliminary result: We find that there exists high relationship between both execution logs generated by unit test and load test. In particular, most of the log events occurring in load tests are covered

in execution logs of unit test. Due to the high coverage, we can confirm that there exists high relationship between unit tests' and load tests' execution logs.

Expected timeline: We plan to finish this part of work before spring 2021.

5 STATE OF THE ART AND PRACTICE

The following research is closely related to our work presented in this dissertation.

Performance regression detection during development.

Prior study has proposed various automated techniques, measurement-based and model-based approaches, to detect performance regression. Measurement-based approaches measure and compare performance metrics between two consecutive versions of a system to detect performance regression [6, 7, 20]. Model-based approaches [1, 5, 13–15] build a detected model for a set of target performance metrics. However, prior research on detection of performance regressions are all designed to be conducted after the system is built and deployed. There is very few works to detect performance regressions at commit level. In our research, we detect performance regressions at commit level.

Leveraging logs to assist performance regression detection in load tests.

There are various approaches to use execution logs to assist performance regression detection in load test. Jiang et al. [10] propose a framework that automatically generates a report to detect and rank potential performance problems and the associated log sequences. Tan et al. [18] present a state-machine view from the execution logs of Hadoop to understand the system behavior and debug performance problems. Syer et al. [17] also propose to combines execution logs and performance metrics to diagnose memory-related performance issues. The existing approaches still need to execute load tests. Our work attempts to predict performance regression using execution logs generated by unit tests, without need to execute load tests.

6 CONCLUSIONS

Performance regressions are not rare instances on large software systems, while existing performance regression detection approaches do not fit well in the widely-adopted process of DevOps. Therefore, this dissertation aims to provide empirical study findings and automated approaches to help practitioners detect performance regressions in the process of DevOps. In particular, to detect performance regressions during development, we study the characteristics of performance regressions at the change level and propose to predict such performance regressions during development. On the other hand, to help detect performance regression using operational data, i.e., without the need of expensive performance testing, we propose approaches to recover field workload and use such workload data to automatically detect performance regressions.

REFERENCES

[1] Peter Bodík, Moisés Goldszmidt, and Armando Fox. 2008. HiLighter: Automatically Building Robust Signatures of Performance Behavior for Small- and Large-Scale Systems. In *Third Workshop on Tackling Computer Systems Problems with Machine Learning Techniques, SysML 2008, December 11, 2008, San Diego, CA, USA, Proceedings.*

[2] Andreas Brunnert, André van Hoorn, Felix Willnecker, Alexandru Danciu, Wilhelm Hasselbring, Christoph Heger, Nikolas Roman Herbst, Pooyan Jamshidi, Reiner Jung, Jóakim von Kistowski, Anne Koziolek, Johannes Kroß, Simon Spinner, Christian Vögele, Jürgen Walter, and Alexander Wert. 2015. Performance-oriented DevOps: A Research Agenda. *CoRR* (2015).

[3] Jinfu Chen and Weiyi Shang. 2017. An Exploratory Study of Performance Regression Introducing Code Changes. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017.* 341–352.

[4] Jinfu Chen, Weiyi Shang, Ahmed E Hassan, Yong Wang, and Jiangbin Lin. 2019. An Experience Report of Generating Load Tests Using Log-recovered Workloads at Varying Granularities of User Behaviour. *The 34th IEEE/ACM International Conference on Automated Software Engineering* (2019).

[5] Ira Cohen, Steve Zhang, Moisés Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. 2005. Capturing, indexing, clustering, and retrieving system history. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles 2005, SOSP 2005, Brighton, UK, October 23-26, 2005.* 105–118.

[6] King Chun Foo, Zhen Ming Jiang, Bram Adams, Ahmed E. Hassan, Ying Zou, and Parminder Flora. 2010. Mining Performance Regression Testing Repositories for Automated Performance Analysis. In *Proceedings of the 10th International Conference on Quality Software, QSIC 2010, Zhangjiajie, China, 14-15 July 2010.* 32–41.

[7] King Chun Foo, Zhen Ming Jiang, Bram Adams, Ahmed E. Hassan, Ying Zou, and Parminder Flora. 2015. An Industrial Case Study on the Automated Detection of Performance Regressions in Heterogeneous Environments. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2.* 159–168.

[8] Christoph Heger, Jens Happe, and Roozbeh Farahbod. 2013. Automated root cause isolation of performance regressions during software development. In *ACM/SPEC International Conference on Performance Engineering, ICPE'13, Prague, Czech Republic - April 21 - 24, 2013.* 27–38.

[9] Michael Hüttermann. 2012. *DevOps for developers.* Apress.

[10] Zhen Ming Jiang, Ahmed E. Hassan, Gilbert Hamann, and Parminder Flora. 2009. Automated performance analysis of load tests. In *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada.* 125–134.

[11] Qi Luo, Denys Poshyvanyk, and Mark Grechanik. 2016. Mining performance regression inducing code changes in evolving software. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016.* 25–36.

[12] Joel Maher. 2017. Mozilla Performance Regressions Policy. https://www.mozilla.org/en-US/about/governance/policies/regressions/

[13] Haroon Malik, Hadi Hemmati, and Ahmed E. Hassan. 2013. Automatic Detection of Performance Deviations in the Load Testing of Large Scale Systems. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13).* IEEE Press, Piscataway, NJ, USA, 1012–1021.

[14] Thanh H. D. Nguyen, Bram Adams, Zhen Ming Jiang, Ahmed E. Hassan, Mohamed N. Nasser, and Parminder Flora. 2012. Automated detection of performance regressions using statistical process control techniques. In *Third Joint WOSP/SIPEW International Conference on Performance Engineering, ICPE'12, Boston, MA, USA - April 22 - 25, 2012.* 299–310.

[15] Thanh H. D. Nguyen, Meiyappan Nagappan, Ahmed E. Hassan, Mohamed N. Nasser, and Parminder Flora. 2014. An industrial case study of automatically identifying performance regression-causes. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India.* 232–241.

[16] Wei Shang, Ahmed E. Hassan, Mohamed N. Nasser, and Parminder Flora. 2015. Automated Detection of Performance Regressions Using Regression Models on Clustered Performance Counters. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, January 31 - February 4, 2015.* 15–26.

[17] D. Syer, Zhen Ming Jiang, Meiyappan Nagappan, Ahmed E. Hassan, Mohamed N. Nasser, and Parminder Flora. 2013. Leveraging Performance Counters and Execution Logs to Diagnose Memory-Related Performance Issues. In *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013.* 110–119.

[18] Jiaqi Tan, Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. 2010. Visual, Log-Based Causal Tracing for Performance Debugging of MapReduce Systems. In *2010 International Conference on Distributed Computing Systems, ICDCS 2010, Genova, Italy, June 21-25, 2010.* 795–806.

[19] E.J. Weyuker and F.I. Vokolos. 2000. Experience with performance testing of software systems: issues, an approach, and case study. *Transactions on Software Engineering* 26, 12 (Dec 2000), 1147–1156.

[20] PengCheng Xiong, Calton Pu, Xiaoyun Zhu, and Rean Griffith. 2013. vPerfGuard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments. In *ACM/SPEC International Conference on Performance Engineering, ICPE'13, Prague, Czech Republic - April 21 - 24, 2013.* 271–282.