Empirical Study of Just-in-Time Prediction of Performance Regression Introducing Change

I. 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 makes 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 [1]. Therefore, performance assurance activities are an essential step in the release cycle of large software systems.

Performance assurance activities aim to identify and eliminate performance regressions in each newly released version. Examples of performance regressions are response time degradation, higher than expected resource utilization and memory leaks. Such regressions may compromise the user experience, increase the operating cost of the system, and cause field failures. The slow response time of the United States' newly rolled-out healthcare.gov illustrates the importance of performance assurance activities before releasing a system. Failure in detecting such regressions would result in significant financial and reputational repercussions.

Prior software quality research typically focus on functional bugs rather performance issues. For example, post-release bugs are often used as code quality measurement and are modelled by statistical modeling techniques in order to understand the relationship between different software engineering activities and code quality [2]. In addition, bug prediction techniques are proposed to prioritize software quality assurance efforts [3]–[5] and assesses the risk of code changes [6]. However, performance regressions are rarely targeted in spite of their importance.

On the other hand, prior study of JIT defect prediction focus more on the risk of commit haved a bug rather than performance regressions. Predicting performance regressions remains a task that is conducted after fact, i.e., after the system is built and deployed in the field or dedicated performance testing environments. However, large amounts of resources are required to detect, locate, understand and fix performance regressions at such a late stage in the development circle; while the amount of required resources would be significantly reduced if developers were notified whether a code change introduces performance regressions during development.

In this research, we perform an empirical study on the JIT prediction of performance regression introducing code changes. By examining the identified performance regression introducing changes, we find that performance regression introducing changes are prevalent during software development. The identified performance regressions are often associated with complex syndrome, i.e., multiple performance metrics have performance regression. In order to build a change risk model to predict JIT performance regression, we combine the basic commit-level measures proposed by Mockus and Weiss [7], which are based on the characteristics of code changes, such as the number of modified subsystems and the purpose of the code change with the performance related measures we added, such as changing conditions and passing expensive parameters.

The rest of this proposal is organized as follows: Section II presents the prior research that is related to this project. Section III presents our subject systems and our approach of predicting performance regression introducing code changes. Section IV presents our research questions. Finally, Section V presents the milestones for the project.

II. Related Work

In this section, we present the related prior research to this paper in three aspects: 1) current state-of-the-art performance regression detection, 2) Prediction of JIT software defect and 3) empirical study on performance.

A. Performance regression detection

A great amount of research has been proposed to detect performance regression.

Ad hoc analysis selects a limited number of target performance counters (e.g., CPU and memory) and performs simple analysis to compare the target counters. Heger et al. [8] present an approach to support software engineers with root cause analysis of the problems. Their approach combines the concepts of regression testing, bisection and call tree analysis to detect performance regression root cause analysis as early as possible.

Pair-wise analysis compares and analyzes the performance metrics between two consecutive versions of a system to detect the problem. Nguyen et al. [9]–[11] conduct a series of studies on performance regressions. Nguyen et al. propose an approach to detect performance regression by using a statistical process control technique called control charts. They construct the control chart and apply it to detect performance regressions and examine the violation ratio of the same performance counter. Malik et al. [12] propose approaches that combine one supervised and three unsupervised algorithms to help performance regression detection. They employ feature selection methods named Principal Component Analysis (PCA) to reduce the dimensionality of the observed performance counter set and validate their approach through a large case study on a real-world industrial software system [13].

Model-based analysis builds a limited number of detected models for a set of target performance counters (e.g., CPU and

memory) and leverages the models to detect performance regressions. Xiong et al. [14] propose a model-driven framework to diagnose the application performance in cloud condition without manual operation. In the framework, it contains three modules consisted of sensor module, model building module and model updating module. It can automatically detect the workload changes in cloud environment and lead to root cause of performance problem. Cohen et al. [15] propose an approach that builds a promising class of probabilistic models (Tree-Augmented Bayesian Networks or TANs) to correlate system level counters and systems average-case response time. Cohen et al. [16] present that performance counters can successfully be used to construct statistical models for system faults and compact signatures of distinct operational problems. Bodik et al. [17] employ logistic regression with L1 regularization models to construct signatures to improve Cohen et al.'s work.

Multi-models based analysis builds multiple models from performance counters and uses the models to detect performance regressions. Foo et al. [18] propose an approach to detect potential performance regression using association rules. They utilize data mining to extract performance signatures by capturing metrics and employ association rules techniques to collect correlations that are frequently observed in the historical data. Then use the change to the association rules to detect performance anomalies. Jiang et al. [19] present two diagnosis algorithms to locate faulty components: RatioScore and SigScore based on component dependencies. They identify the strength of relationships between metric pairs by utilizing an information-theoretic measures and track system state based on in-cluster entropy. A significant change in the in-cluster entropy is considered as a sign of a performance fault. Shang et al. [20] propose an approach that first clusters performance metric based on their correlation. Each cluster of metrics is used to build statistical model to detect performance regressions.

Prior research on performance regressions are designed to be conducted after the system is built and deployed in either performance testing environments or user environments. In this paper, we explore performance regression at commit level, i.e., when the performance regressions are introduced into the software.

B. Prediction of JIT software defect

C. Empirical studies on performance

Empirical studies are conducted in order to study performance issues. Jin et al. [21] studies 109 real world performance issues that are reported from five open source software. Based on the studied 109 performance bugs, Jin et al. [21] develop an automated tool to detect performance issues. Zaman et al. [22], [23] conducted both qualitative and quantitative studies on performance issues. They find that developers and users face problems in reproducing performance bugs. More time is spent on discussing performance bugs than other kinds of bugs. Huang et al. [24] studied real world performance issues and based on the findings, they propose an approach called performance risk analysis (PRA), to improve the efficiency of performance regression testing.

Luo et al. [25] propose a recommendation system, called PerfImpact, to automatically identify code changes that may potentially be responsible for performance regression between two releases. Their approach searches for input values that expose performance regressions and compare execution traces between two releases of a software to identify problematic code changes. Hasan et al. [26] create energy profiles as a performance measurement for different Java collection classes. They find that the energy consumption can have large difference depending on the operation.

Prior studies on performance typically are based on either limited performance issue reports or release of the software. However, the limit amount of issue reports and releases of the software hides the prevalence of performance regressions. In our paper, we evaluate performance at commit level. Therefore, we are able to identify more performance regressions and are able to observe the prevalence of performance regression introducing changes in development.

III. Approach

In this section we will explain our methodology in more detail. At first we depict our subject, including the open-source projects we choose and the experimental environment we set up. Then we present each step of our approach.

A. Subject systems

We choose two open-source projects, *Hadoop* and *RxJava* as the subject systems of our case study. *Hadoop* [27] is a distributed system infrastructure developed by the Apache Foundation. *Hadoop* performs data processing in a reliable, efficient, high fault tolerance, low cost and scalable manner. We choose *Hadoop* since it is highly concerned with its performance and has been studied in prior research in mining performance data [28]. *RxJava* is a library for composing asynchronous and event-based programs by using observable sequences and it carries the JMH benchmarks test options. *RxJava* is a *Java* VM implementation of reactive extensions. *RxJava* provides a slew of performance micro-benchmarks, making it an appropriate subject for our study. The overview of the two studied systems is shown in Table I.

TABLE I: Overview of our subject systems.

Subjects	Version	Total lines of code (K)	No. of files
Hadoop	2.7.2	1167	6,371
Hadoop	2.7.3	1568	6,439
RxJava	2.0.0	164	1,107
RxJava	2.0.1	242	1,513
RxJava	2.0.2	243	1,524
RxJava	2.0.3	244	1,524
RxJava	2.0.4	244	1,526

IV. Case Study

In this section, we perform an exploratory study on the extracted performance regressions from our subject systems (*Hadoop* and *RxJava*). Our study aims to answer two research questions. For each research question, we present the motivation of the question, the approach that we use to answer the question, the results to the question and we discuss the results.

RQ1: How well can we predict the existence of performance regression introducing change?

Motivation

Prior research has conducted empirical studies on performance bugs [21], using the reported performance bugs in issue reports (like JIRA issues). However, there may exist much more performance issues, such as performance regressions, that are not reported as JIRA issues. On the other hand, we evaluate performance of on each code commit instead of depending on JIRA issues. Intuitively, we may uncover instances of performance regressions that are not reported, and are not be able to investigated using the approach of prior studies. Therefore, in this research question, we start off by examining how prevalence are detected performance regression introducing changes. If we could not identify performance regressions in the subject systems, our study would be of less value to the community.

Approach

With the approach presented in Section III, we obtain the results of performance evaluation for every commit in our subject systems. In order to study the prevalence of performance regression introducing changes, we first examine whether each commit would cause any test case to complete with a significantly longer response time. In particular, we only consider a test having performance regression if the response time is statistically significantly longer and the effect size is non-trivial. In addition, we use the effect sizes to measure the magnitude of the performance regression in each test.

Sometimes performance regressions may not cause impact on response time but rather cause a higher resource utilization. The high resource utilization, although may not directly impact user experience, may cause extra cost when deploying, operating and maintaining the system, with lower scalability and reliability. For example, systems that are deployed on cloud providers (like Microsoft Azure) may need to choose virtual machines with higher specification for higher resource utilization. Moreover, a software release with a higher memory usage is more prone to crashes from memory leaks. Therefore, we also use the physical metrics, i.e., CPU usage, Memory usage, I/O read and I/O write, as measurements of performance regressions.

In order to understand whether each performance metrics can provide extra information to others, we also calculate the Pearson correlation between the effect sizes of performance regressions calculated using different metrics. Therefore, we would understand whether we can use a smaller set of metrics to identify performance regressions.

RQ2: How well can we predict the magnitude of performance regression introducing change?

Motivation

In RQ1, we find that there exist prevalent performance regressions that are introduced by code changes. If we can understand what cause the introduction of these performance regressions, we may provide guidance or automated tooling support for developer to prevent the regressions during code change.

Approach

We follow two steps in our approach to discover the reasons of introducing performance regressions. First, we investigate the high-level context when these performance regressions are introduced. We identify the type of issues as the context (fixing a bug or developing new features) that are related to the performance regression introducing changes.

Second, we would like to know the code level root-causes (e.g., what kind of code change) of why the performance regressions are introduced. In particular, for each commit, we manually examine all the code changes and the corresponding test cases where performance regression are identified. We follow an iterative approach to identify the root-causes that the code change introduces performance regression, until we could not find any new reasons.

RQ3: What are the important measures of performance regression introducing change?

V. Milestones

In this section we present the milestones for our project.

References

[1] E. Weyuker and F. Vokolos, "Experience with performance testing of software systems: issues, an approach, and case study," *Transactions on Software Engineering*, vol. 26, no. 12, pp. 1147–1156, Dec 2000.

[2] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 78–88.

[3] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, May 2007, pp. 9–9.

[4] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 284–292.

[5] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 452–461.

[6] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, June 2013.

[7] A. Mockus and D. M. Weiss, "Predicting risk of software changes," *Bell Labs Technical Journal*, vol. 5, no. 2, pp. 169–180, 2000.

[8] C. Heger, J. Happe, and R. Farahbod, "Automated root cause isolation of performance regressions during software development," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 27–38.

[9] T. H. Nguyen, B. Adams, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, "Automated detection of performance regressions using statistical process control techniques," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '12. New York, NY, USA: ACM, 2012, pp. 299–310.

[10] T. H. D. Nguyen, B. Adams, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, "Automated verification of load tests using control charts," in *2011 18th Asia-Pacific Software Engineering Conference*, Dec 2011, pp. 282–289.

[11] T. H. D. Nguyen, M. Nagappan, A. E. Hassan, M. Nasser, and P. Flora, "An industrial case study of automatically identifying performance regression-causes," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 232–241.

[12] H. Malik, H. Hemmati, and A. E. Hassan, "Automatic detection of performance deviations in the load testing of large scale systems," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 1012–1021.

[13] H. Malik, Z. M. Jiang, B. Adams, A. E. Hassan, P. Flora, and G. Hamann, "Automatic comparison of load tests to support the performance analysis of large enterprise systems," in *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*. IEEE, 2010, pp. 222–231.

[14] P. Xiong, C. Pu, X. Zhu, and R. Griffith, "vperfguard: An automated model-driven framework for application performance diagnosis in consolidated cloud environments," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 271–282.

[15] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control." in *OSDI*, vol. 4, 2004, pp. 16–16.

[16] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," in *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, ser. SOSP '05. New York, NY, USA: ACM, 2005, pp. 105–118.

[17] P. Bodík, M. Goldszmidt, and A. Fox, "Hilighter: Automatically building robust signatures of performance behavior for small-and large-scale systems." in *SysML*, 2008.

[18] K. C. Foo, Z. M. Jiang, B. Adams, A. E. Hassan, Y. Zou, and P. Flora, "Mining performance regression testing repositories for automated performance analysis," in *Quality Software (QSIC), 2010 10th International Conference on*. IEEE, 2010, pp. 32–41.

[19] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. Ward, "Automatic fault detection and diagnosis in complex software systems by information-theoretic monitoring," in *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*. IEEE, 2009, pp. 285–294.

[20] W. Shang, A. E. Hassan, M. Nasser, and P. Flora, "Automated detection of performance regressions using regression models on clustered performance counters," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '15. New York, NY, USA: ACM, 2015, pp. 15–26.

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

[22] S. Zaman, B. Adams, and A. E. Hassan, "Security versus performance bugs: a case study on firefox," in *Proceedings of the 8th working conference on mining software repositories*. ACM, 2011, pp. 93–102.

[23] S. Zaman, B. Adams, and Hassan, "A qualitative study on performance bugs," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, ser. MSR '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 199–208.

[24] P. Huang, X. Ma, D. Shen, and Y. Zhou, "Performance regression testing target prioritization via performance risk analysis," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 60–71.

[25] Q. Luo, D. Poshyvanyk, and M. Grechanik, "Mining performance regression inducing code changes in evolving software," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 25–36.

[26] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of java collections classes," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 225–236.

[27] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[28] M. D. Syer, W. Shang, Z. M. Jiang, and A. E. Hassan, "Continuous validation of performance test workloads," *Automated Software Engineering*, vol. 24, no. 1, pp. 189–231, 2017.