Addressing Performance Regressions in DevOps: Can We Escape from System Performance Testing?

Lizhi Liao
Concordia University
Montreal, Canada
l_lizhi@encs.concordia.ca

***Abstract*—Performance regression is an important type of performance issue in software systems. It indicates that the performance of the same features in the new version of the system becomes worse than that of previous versions, such as increased response time or higher resource utilization. In order to prevent performance regressions, current practices often rely on conducting extensive system performance testing before releasing the system into production based on the testing results. However, faced with a great demand for resources and time to perform system performance testing, it is often challenging to adopt such approaches to the practice of fast-paced development and release cycles, e.g., DevOps. This thesis focuses on addressing software performance regressions in DevOps without relying on expensive system performance tests. More specifically, I first propose a series of approaches to helping developers detect performance regressions and locate their root causes by only utilizing the readily-available operational data when the software system is running in the field and used by real end users. I then leverage small-scale performance testing and architectural modeling to estimate the impact of source code changes on the end-to-end performance of the system in order to detect performance regressions early in the software development phase. Through various case studies on open-source projects and successful adoptions by our industrial research collaborator, we expect that our study will provide helpful insights for researchers and practitioners who are interested in addressing performance regressions in DevOps without expensive system performance testing.**

***Index Terms*—performance regression, performance regression root cause, field testing, performance modeling, performance engineering**

I. Introduction

Performance is an essential quality attribute of software systems. It measures how effective a software system is with various metrics such as response time, throughput, or CPU utilization. A prior study [22] indicates that compared to functional issues, performance issues are more likely to cause field failures in large-scale software systems. As a common and important type of performance issue, performance regressions are the circumstances where the new version of a system still functions correctly, but offers a worse user experience (e.g., higher response time) and/or consumes extra resources (e.g., memory leak) compared to previous versions. Performance regressions may decrease user satisfaction, increase operating costs, and cause field failures [7]. This is especially critical for large-scale software systems, such as Amazon or Netflix, since they need to continuously provide high-quality services to millions of clients across the globe, and performance regressions in the system can cause serious financial and reputation losses [3], [20]. In addition, more and more software systems adopt DevOps practices that integrate development (Dev) and operations (Ops) for continuous and fast delivery (e.g., a new version is released every few hours) [24]. However, it also poses the challenges of addressing performance regressions prior to the new release of such systems, since performance assurance activities often require a considerable amount of time to conduct. Therefore, how to address performance regressions in DevOps, including detecting the existence of performance regressions and locating their root causes, is an important yet challenging task to prevent the new version of the system from performing worse than previous versions.

Current practices that address performance regressions often rely on running system performance tests in the in-house testing environment prior to the deployment of every new version of a software system. There has been extensive prior research [1], [13], [14], [17]–[19], [21] that proposes various techniques, such as control charts, associations rules, and statistical models, that compare and analyze the performance testing results (e.g., performance metrics) in order to detect performance regressions and locate the root causes. However, system performance testing is often expensive in resources and time, making it hard to adopt in the DevOps process. It also requires a similar or even the same testing workload between performance testing different software versions, while the real-world workload is constantly varying and may even be completely different over time. Furthermore, to avoid the expensive performance testing on the entire system, unit testing [4], [8], [12] and field data [2], [9], [15], [23] are also leveraged in prior studies to address performance regressions. Nevertheless, unit testing-based approaches can only be aware of the performance of separate small-scale components, while existing field data-based approaches often yield too coarse-grained results (e.g., only at the service level) to provide developers with enough information in fixing the regressions.

Research hypothesis: By leveraging the abundant information during software development and field operations, I hypothesize that we can provide effective and timely support to developers and operators in addressing performance regressions in DevOps without expensive system performance testing.

As summarized in Figure 1, addressing performance re-

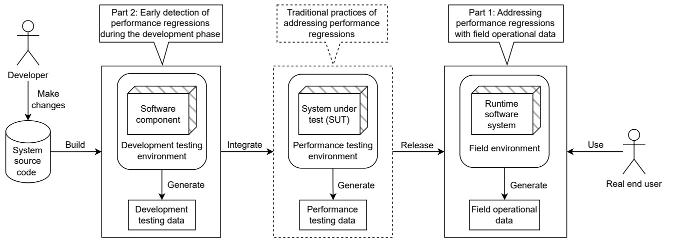
Part 2: Early detection of performance regressions during the development phase
Traditional practices of addressing performance regressions
Part 1: Addressing performance regressions with field operational data
Developer
Make changes
System source code
Build
Software component
Development testing environment
Integrate
System under test (SUT)
Performance testing environment
Release
Runtime software system
Field environment
Use
Real end user
Generate
Development testing data
Generate
Performance testing data
Generate
Field operational data

Fig. 1. The overall process of addressing performance regressions during the software development and release cycle

gressions may be related to different phases of the software development and release cycle. Faced with the challenges of traditional practices that rely on time- and resource-consuming system performance testing, I hypothesize that by analyzing software development and field operational data, we can provide automated solutions to tackle the challenges and assist developers and operators in detecting performance regressions and locating the root causes in DevOps.

In this thesis, we first attempt to address performance regressions directly based on the operational data when the system is deployed in the field (cf. Part 1 in Figure 1). In particular, we will leverage black-box machine learning models to automatically detect performance regressions when the system is under field operations. We will then propose a technique to locate the root causes of performance regressions by utilizing the readily-available operational data when the software system is directly deployed in the field. Finally, we will adapt existing performance analytic techniques in a real-life database-centric system from our industrial research collaborator. We also attempt to detect performance regressions by directly analyzing the data collected during the software development phase (cf. Part 2 in Figure 1). In particular, we will propose an approach that predicts the impact of source code changes on the end-to-end performance of the system to detect the existence of performance regressions as early as during the development phase. I expect to finish the work for this thesis by the end of 2023.

Paper organization. The rest of the paper is structured as follows. Section II and Section III respectively present our proposed approaches to assisting developers and operators in addressing performance regressions with field operational data and early detection of performance regressions during the development phase. Section IV surveys prior research related to this thesis. Finally, Section V concludes our work.

II. Addressing Performance Regressions with Field Operational Data

For large-scale systems delivered at a fast pace (e.g., DevOps), it is often challenging to conduct system performance testing prior to a new release due to the great demand for resources and time. This is especially the case for our industrial research collaborator. In particular, the stakeholders would like to be aware of any performance regressions introduced in a new version of the system. However, the industrial system follows a fast-paced DevOps process where the system has a new version every two weeks. Such a short release cycle brings challenges in detecting performance regressions and locating the root causes. On the other hand, the software operational data generated in the field provides abundant information about the performance of a software system and its runtime behaviors. Therefore, in the first part of our research, we propose a series of automated approaches to helping developers and operators first detect performance regressions and then locate performance regression root causes directly based on analyzing field operational data without the need for system performance testing, and finally, we adapt the existing performance analytic techniques in a real-life database-centric system.

A. Detecting Performance Regressions in the Field Operations

Problem: System performance testing with predefined test workloads is commonly employed in an in-house (non-production) testing environment before delivering the system into production to detect performance regressions. However, such a practice can be extremely expensive since it essentially requires costly resources, complex configurations, and extended time. Furthermore, it is also hard to resemble the field workloads in the predefined test suites as the real-life workloads can constantly vary over time.

Our proposed solution: Inspired by some existing software testing techniques, such as A/B testing or canary releasing, which rely on the end user's data in the field for software quality assurance, in this work, we propose to directly detect performance regressions based on the field operational data while a software system is deployed and running in the field. In particular, we utilize sparsely-sampled performance metrics and readily-available execution logs from the field, which only add negligible performance overhead to the runtime system. We then leverage various black-box machine learning and deep learning techniques (i.e., Linear Regression, Random Forest, XGBoost, RNN, LSTM, and CNN) to capture the relationship between the runtime activities (i.e., workloads) that are recorded in the execution logs and their performance under such activities. Afterward, we analyze and compare the black-box performance model that describes the current version of a system and the model that describes an earlier version of the same system to identify the existence of performance regressions between these two versions. Since we utilize field operational data from end users and the workloads from two versions are often inconsistent, we finally perform evaluation experiments on both open-source and industrial systems to study the effectiveness of using black-box performance models to detect performance regressions.

Results: We find that our black-box performance models can effectively model the performance of the studied systems with a mean relative error (MRE) as low as 2.11%. The experiment results also show that our approach can successfully identify the performance regressions between the old and new versions in the field operations with a significant difference in the statistical analysis results (i.e., with a magnitude of medium or large in effect size).

Expected timeline: This work is completed and published at EMSE [10].

B. Locating Performance Regression Root Causes in the Field Operations

Problem: Locating the root causes of software performance regressions remains a challenging yet vital task for developers due to the considerable effort, time, and expertise required. Prior research has proposed various approaches that utilize system performance testing, unit testing, or field data to locate performance regression root causes. However, existing approaches are often resource-intensive and time-consuming, pay no attention to the impact of large and varying system workloads, or can only locate too coarse-grained root causes.

Our proposed solution: To bridge the gap, we propose an approach to assisting developers in locating performance regression root causes in the field operations of web-based systems. We first extract the web-access logs recorded by web servers and the performance metrics collected by performance monitoring tools. Afterward, we preprocess the data and construct performance models that capture the performance of the system and its runtime activities for the old and the new versions of the systems respectively. The intuition of our approach is that the root cause of performance regressions is related to the deviation between these two performance models. Therefore, we then calculate the deviation of modeling errors of these two performance models and build a linear regression model to explain the relationship between the appearance of web requests and the deviation of modeling errors. We only keep the independent variables (i.e., web requests) that have a statistically significant effect on the output of the model (i.e., deviation of modeling errors) and rank them by the effect. We then leverage static source code analysis techniques to extract the call graph for each target web request. Finally, we locate the code changes that change any methods along the extracted call graph and provide them to developers to assist in fixing regressions. Our evaluation considers three open-source projects and one commercial system.

Results: From the results, we observe that our approach can distinguish between system versions with and without performance regressions by leveraging R^2 of the constructed linear regression model. For example, in the results of the *OpenMRS* subject, the R^2 of the version without injected regressions is only 0.14, while for all other versions with injected regressions, the lowest R^2 is 0.50. The experiment results also present that by applying our approach, there is usually a large difference between the effect of web requests with performance regressions and the highest effect of web requests without performance regressions. Finally, we find that for all open-source subjects, the web requests with performance regressions are always ranked in the first place and we can also successfully locate the corresponding code changes.

Expected timeline: This work is completed and published at TSE [11].

C. Adapting Existing Performance Analytic Techniques in a Real-Life Database-Centric System

Problem: Many large-scale systems embrace the database-centric design to cope with the ever-increasing complexity, scale, and amount of data. There are several studies and techniques proposed to diagnose performance issues and improve the performance of traditional software systems. However, directly applying existing techniques in large-scale database-centric systems is often challenging and may not perform well due to the unique nature of database-centric systems, for example, a major part of the business logic and calculations reside in the database rather than in the application server.

Our proposed solution: To assist developers in the performance assurance of large-scale database-centric systems, we share our industrial experience of adapting the existing performance analytic techniques in a real-life database-centric system from our industrial research collaborator. By considering both the database and the surrounding components, our solution aims to provide multiple-aspect automated performance analysis, including detecting performance issues within one version, between two versions, and root cause analysis, to improve the system performance while minimizing the need for manual efforts and expertise. During the engineering process of adapting the existing performance analytic techniques in the industrial setting, we encountered many challenges in terms of design and development, and then we proposed solutions to resolve them. In particular, we first propose to record system performance and runtime behaviors to have a comprehensive understanding of the running system from the ground up. Based on the rich information, we then aim to provide automated performance analysis of the entire system including performance analysis within one version and performance analysis between two versions. In order to help developers locate the root causes of performance issues, we also propose to add version management to the database code changes in addition to the application code changes. We then propose to utilize static code and string analysis techniques to connect the problematic database activities (e.g., SQL queries) to the corresponding web application code and connect problematic web application activities (e.g., web requests) to the corresponding database code. Finally, we provide these root causes to developers for further investigation.

Preliminary results: Our developed performance analytic techniques have been adopted and used by our industrial research collaborator to ensure the performance of a large-scale database-centric system on a daily basis. We have assisted developers from our industrial research collaborator to successfully detect several real-life performance issues that were missed before. We believe that our experience in adapting existing performance analytic techniques to database-centric systems can provide valuable insights to interested software practitioners and researchers.

Expected timeline: This work is expected to be completed before the spring of 2023.

III. Early Detection of Performance Regressions during the Development Phase

Software performance regressions are often addressed late in the software development and release cycle, for instance, after the system is built, integrated, or even released. Not only does this make it laborious for developers to detect, locate, and fix performance regressions, but it can also lead to potential adverse effects on users and companies. Therefore, in the second part of our research, our goal is to provide developers with an early impression of how the source code changes in specific components are related to the presence of performance regressions of the entire software system.

A. Bridging Local Performance Data and Architectural Models: A Performance Regression Detection Approach

Problem: During a development iteration, developers may introduce many source code changes to implement new features or fix bugs, some of which may degrade the system performance. Inspired by this, prior research advocates using tests on a smaller scale, such as micro-performance benchmarks or functional tests, to detect performance regressions. Although small-scale testing and profiling of individual software components are easy to operate and time efficient, they often suffer from the inability to take the end-to-end performance and system workloads into consideration.

Our proposed solution: To tackle the problem, we plan to propose an approach to estimating the impact of component-level performance deviation on the end-to-end performance of the entire system to detect performance regressions as early as during the software development phase. In particular, we first extract component-level call graphs with related performance information by running performance unit tests (e.g., Java Microbenchmark Harness (JMH)) of specific components and extract system-level call graphs with related performance information by monitoring and analyzing the behaviors of the running system in terms of both structural and performance aspects. Based on this information, we then build an analytical model (e.g., Queueing Petri Nets) that can predict the end-to-end performance of the system. Afterward, we utilize statistically rigorous approaches to identify the components that suffer from degraded performance and calculate their impact on the corresponding architecture-level performance information by analyzing the component-level and system-level call graphs previously extracted. Finally, we propagate such impact into the constructed performance model of the system and leverage the model to predict the end-to-end performance deviation to detect performance regressions of the entire software system.

Our proposed evaluation: We will apply our approach to various prevalent open-source applications with both synthetic and real-life performance regressions to evaluate the effectiveness of our approach in detecting performance regressions during the development phase.

Expected timeline: This work is expected to be completed before the summer of 2023.

IV. State of the art and practice

In this section, we discuss the current research that is closely related to our work in this thesis.

Detecting performance regressions. Previous research has proposed various approaches to detecting performance regressions. Nguyen et al. [16], [18] leverage a statistical process control technique named control charts that compare each performance metric from the old and new versions pairwise to automatically detect performance regressions. Foo et al. [6] extract association rules between multiple performance metrics collected during system performance testing and identify performance regressions by the change to the mined association rules between the two versions. Shang et al. [19] propose to group a great number of performance metrics into multiple clusters and then build regression models for each cluster to detect performance regressions. However, prior research relies on the data generated from resource- and time-consuming system performance testing with predefined or fixed workloads. Our work attempts to automatically detect performance regressions by leveraging the development and field operational data that are more affordable and accessible.

Locating performance regression root causes. There are several studies that analyze different types of data to locate performance regression root causes. The first type [14], [17], [21] analyzes the system performance testing results (e.g., performance metrics) across versions of software systems to locate performance regression root causes. The second type [4], [8], [12] proposes to adopt unit tests and combine the testing results with static or dynamic source code analysis techniques to locate the root cause. The last type [2], [5], [23] utilizes system runtime information (e.g., execution logs and performance metrics) collected directly from end-user operations in the field to locate performance regression root causes. However, system performance tests usually require extensive resources and a long time to execute, unit tests-based approaches only consider the performance of small-scale components and cannot take the impact of large and varying system workloads into consideration, and the located root causes generated by the existing field-data based approaches are often too coarse-grained (e.g., only at the service level) to provide enough help for developers to fix the regressions. In our work, we aim to locate fine-grained performance regression root causes (e.g., source code changes) under continuously varying workloads without the need for executing expensive system performance tests.

V. Conclusion

Performance regressions are common and vital in large-scale software systems. However, existing practices in addressing performance regressions often do not fit well in the fast-paced process of DevOps. In this thesis, we aim to provide a series of automated approaches and share our industrial adoption experience to assist developers in addressing performance regressions in DevOps by leveraging the data collected during the software development and field operations, rather than relying on running expensive system performance tests. Our current progress and future plans would provide insights and support to benefit practitioners and researchers in conducting performance assurance activities during DevOps without expensive system performance testing. I expect to finish the work for this thesis by the end of 2023.

References

[1] J. P. S. Alcocer, A. Bergel, and M. T. Valente, "Learning from source code history to identify performance failures," in *Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering, ICPE 2016, Delft, The Netherlands, March 12-16, 2016*. ACM, 2016, pp. 37–48.

[2] E. R. Altman, M. Arnold, S. Fink, and N. Mitchell, "Performance analysis of idle programs," in *Proceedings of the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA*. ACM, 2010, pp. 739–753.

[3] A. B. Bondi, *Foundations of software and system performance engineering: process, performance modeling, requirements, testing, scalability, and practice*. Pearson Education, 2014.

[4] J. Chen and W. Shang, "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*. IEEE Computer Society, 2017, pp. 341–352.

[5] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," *International Conference on Autonomic Computing, 2004. Proceedings.*, pp. 36–43, 2004.

[6] K. C. Foo, Z. M. Jiang, B. Adams, A. E. Hassan, Y. Zou, and P. Flora, "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*, 2015, pp. 159–168.

[7] B. Gregg, *Systems performance: enterprise and the cloud*. Pearson Education, 2014.

[8] C. Heger, J. Happe, and R. Farahbod, "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*. ACM, 2013, pp. 27–38.

[9] H. Jayathilaka, C. Krintz, and R. Wolski, "Performance monitoring and root cause analysis for cloud-hosted web applications," in *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*. ACM, 2017, pp. 469–478.

[10] L. Liao, J. Chen, H. Li, Y. Zeng, W. Shang, J. Guo, C. Sporea, A. Toma, and S. Sajedi, "Using black-box performance models to detect performance regressions under varying workloads: an empirical study," *Empir. Softw. Eng.*, vol. 25, no. 5, pp. 4130–4160, 2020.

[11] L. Liao, J. Chen, H. Li, Y. Zeng, W. Shang, C. Sporea, A. Toma, and S. Sajedi, "Locating performance regression root causes in the field operations of web-based systems: An experience report," *IEEE Transactions on Software Engineering*, pp. 1–1, 2021.

[12] 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, MSR 2016, Austin, TX, USA, May 14-22, 2016*. ACM, 2016, pp. 25–36.

[13] 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.

[14] D. Maplesden, K. von Randow, E. D. Tempero, J. G. Hosking, and J. C. Grundy, "Performance analysis using subsuming methods: An industrial case study," in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2*. IEEE Computer Society, 2015, pp. 149–158.

[15] V. Nair, A. Raul, S. Khanduja, V. Bahirwani, S. Sellamanickam, S. S. Keerthi, S. Herbert, and S. Dhulipalla, "Learning a hierarchical monitoring system for detecting and diagnosing service issues," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*. ACM, 2015, pp. 2029–2038.

[16] 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*, 2011, pp. 282–289.

[17] 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*, 2014, p. 232–241.

[18] 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*, 2012, pp. 299–310.

[19] 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*, 2015, p. 15–26.

[20] C. U. Smith and L. G. Williams, *Performance solutions: a practical guide to creating responsive, scalable software*. Addison-Wesley Reading, 2002.

[21] M. D. Syer, Z. M. Jiang, M. Nagappan, A. E. Hassan, M. N. Nasser, and P. Flora, "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*. IEEE Computer Society, 2013, pp. 110–119.

[22] E. J. Weyuker and F. I. Vokolos, "Experience with performance testing of software systems: Issues, an approach, and case study," *IEEE Trans. Software Eng.*, vol. 26, no. 12, pp. 1147–1156, 2000.

[23] 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*, 2013, p. 271–282.

[24] L. Zhu, L. Bass, and G. Champlin-Scharff, "Devops and its practices," *IEEE Softw.*, vol. 33, no. 3, pp. 32–34, 2016.