2016

Automatic Performance Testing using Input-Sensitive Profiling

Qi Luo
William & Mary

Automatic Performance Testing using Input-Sensitive Profiling

Qi Luo
Department of Computer Science
College of William and Mary
Williamsburg, VA, USA
qluo@cs.wm.edu

ABSTRACT

During performance testing, software engineers commonly perform application profiling to analyze an application's traces with different inputs to understand performance behaviors, such as time and space consumption. However, a non-trivial application commonly has a large number of inputs, and it is mostly manual to identify the specific inputs leading to performance bottlenecks. Thus, it is challenge is to automate profiling and find these specific inputs. To solve these problems, we propose novel approaches, FOREPOST, GA-Prof and PerfImpact, which automatically profile applications for finding the specific combinations of inputs triggering performance bottlenecks, and further analyze the corresponding traces to identify problematic methods. Specially, our approaches work in two different types of real-world scenarios of performance testing: i) a single-version scenario, in which performance bottlenecks are detected in a single software release, and ii) a two-version scenario, in which code changes responsible for performance regressions are detected by considering two consecutive software releases.

CCS Concepts

•Software and its engineering → Software performance; Software testing and debugging;

Keywords

Input-sensitive profiling, performance testing, machine learning algorithms, genetic algorithms, change impact analysis

1. INTRODUCTION

Performance testing is one of the most important activities for engineers to identify inadequate performance behaviors, such as longer execution time and/or lower throughput, for an Application Under Testing (AUT) [17]. During testing, engineers commonly utilize profiling tools to collect dynamic information (e.g., execution time) with some inputs for linking AUT performance behaviors with inputs. After selecting the specific inputs likely to trigger bottlenecks, engineers further analyze the corresponding traces to locate the problematic methods. Specifically, they need to consider two real-world scenarios during testing: i) a single-version scenario, in which bottlenecks are identified in a single software version, and ii) a two-version scenario, in which code changes responsible for performance regressions are identified in two consecutive releases.

For a non-trivial AUT, application profiling is mostly manual and time-consuming due to the large body of combinations of inputs. A big challenge of profiling is to automate the profiling process with the specific input data to trigger bottlenecks. Furthermore, it is also difficult to analyze execution traces to deeply understand the behaviors for identifying the causes of the exposed performance bottlenecks. To solve these problems, we proposed several novel approaches (e.g., FOREPOST, GA-Prof, and PerfImpact) to automatically profile AUTs for exposing performance bottlenecks and further identifying the problematic methods in two performance testing scenarios [17, 26, 15, 16, 18, 9, 14].

FSE'16, November 13–18, 2016, Seattle, WA, USA
ACM. 978-1-4503-4218-6/16/11...$15.00
http://dx.doi.org/10.1145/2950290.2983975

2. RELATED WORK

A large body of research work has focused on improving performance testing [21, 27, 1, 28, 24, 8]. Coppa *et al.* introduce a profiling idea to automatically measure the relationship between performance scalability and input sizes [6]. Zhang *et al.* introduce a mixed symbolic execution approach to generate tests with worsen time and memory consumption [28]. Nguyen *et al.* use control charts to identify the specific tests with worsen performance in a new release [19, 20]. However, little effort has been put on investigating the problematic methods leading to performance bottlenecks. Jin *et al.* performed an empirical study on 109 real performance bugs to extract efficiency-related rules for performance problem detection [12]. Huang *et al.* built a static model to measure the risk of each commit for targeting the ones responsible for the performance problems [10]. However, they are only applicable for specific patterns of performance problems and utilize static analysis to understand performance behaviors, which is prone to be time-consuming. Conversely, our approaches are aimed at selecting specific inputs to expose performance bottlenecks, and analyzing corresponding execution traces to identify problematic methods for those input-sensitive performance bottlenecks.

3. APPROACHES

This section shows approaches that automate input-sensitive profiling for performance testing in two scenarios.

3.1 A Single-Version Scenario

We propose FOREPOST and its alternative version FOREPOST$_{RAND}$ [17, 16], which rely on a Machine Learning algorithm (ML), RIPPER [2], to extract rules mapping performance behaviors to inputs. These rules are used to guide input selection for automating profiling and exposing bottlenecks. Furthermore, FOREPOST and FOREPOST$_{RAND}$ utilize Independent Component Analysis (ICA) [11, 7] to analyze the corresponding traces for identifying bottlenecks. Initially, random inputs are selected and a profiling tool is used to collect trace information (e.g., execution time) for each combination of inputs. All traces are classified into two groups, "good" and "bad". Traces with longer execution times are marked as good, which are "good" to expose bottlenecks. Traces with shorter execution times are marked as bad, which are "bad" to expose bottlenecks. Based on the classified traces, ML extracts rules for describing performance behaviors with the corresponding inputs. FOREPOST uses these rules to choose inputs exposing performance bottlenecks (FOREPOST$_{RAND}$ also involves some random inputs), and start the profiling process for next iteration. Once there is no new rules extracted, profiling process is terminated. ICA is used to understand method's contributions to different performance behaviors and identify the problematic methods. Our hypothesis is that the methods with significant contributions to good traces but less/no contributions to bad traces are marked as bottlenecks.

In addition, we introduce PRESTO, which deploys FOREPOST in the cloud to help developers improve provisioning strategies guiding the cloud to (de)allocate resources for AUTs [9]. It first utilizes FOREPOST to build performance behavior models for AUTs, and then performs sensitivity analysis into provisioning strategies based on behavior models to obtain the strategies that concisely describe the relationship between inputs and resources (e.g., CPU, memory). These strategies are used to recommend requested sources to engineers who maintain AUTs' quality of service.

While FOREPOST finds specific inputs for automating profiling and identifies performance bottlenecks, it has been shown to miss some bottlenecks due to the limitations of extracted rules. FOREPOST only selects specific inputs based on extracted rules, thus, it is likely to focus on some locally hot paths but fails to explore the whole AUT comprehensively. To explore input data space as whole, we propose GA-Prof [26], which uses Genetic Algorithms (GAs) to select the specific input data likely to expose bottlenecks, automating profiling. The intuition behind this approach is mapping the selection of input data to a search and optimization problem. In GA-Prof, the instrumented AUT is running with initial inputs, and profiling information is collected and analyzed. After analyzing profiling information, GAs marks the inputs with longer elapsed execution times as fitter ones, and selects these inputs to create inputs for the next generation. GA process will be terminated when the pre-defined termination criteria are satisfied. The inputs selected by GA-Prof in the last generation are considered as the ones exposing bottlenecks.

3.2 A Two-Version Scenario

While FOREPOST and GA-Prof are able to identify bottlenecks in a single-version scenario of performance testing, they are not suitable to expose performance regressions in a two-version scenario and further locate the problematic code changes leading to the exposed performance regressions. Thus, we propose PerfImpact [18], which uses GAs to select test input data with worsen performance behaviors in a newly released version as compared to the behaviors in a previous version, and further utilizes Change Impact Analysis (CIA) [13] to analyze change's impact on performance degradation for identifying the problematic ones. First, inputs are selected randomly and sent to two versions of AUT independently. After profiling tool collects traces for each version, we obtain execution times of two versions for each combination of inputs. We hypothesize that the regression-exposing inputs have longer execution times in a new version as compared to the times in an old version.Thus, PerfImpact calculates the difference of execution times between two versions for each combination of inputs as its fitness value, and selects the ones with larger differences to create new inputs for the next generation, exposing performance regressions.Then, CIA is used to analyze corresponding execution traces to understand a method's impact on performance behaviors. We obtain an impact set for each code change, which contains a set of methods dynamically impacted by the code change. The code changes whose impact sets contain more methods with performance degradations in new version are marked as problematic ones.

4. RESULTS AND CONTRIBUTIONS

We evaluated FOREPOST, FOREPOST$_{RAND}$ and GA-Prof on one commercial software and three open-source applications [8, 17, 26]. The experimental results show that FOREPOST and FOREPOST$_{RAND}$ are able to select the inputs with longer time for exposing bottlenecks as compared to random inputs, and effectively identify real-world bottlenecks confirmed by developers. GA-Prof has been shown to be able to locate more performance bottlenecks (i.e., 5.6 bottlenecks) as compared to FOREPOST (i.e., 2 bottlenecks) [26]. The potential reason behind this is that GA-Prof is able to search test input space as a whole for finding inputs exposing performance bottlenecks, while FOREPOST only focuses on the specific ones based on extracted rules, missing some computationally intensive execution paths. Furthermore, we evaluated PerfImpact on multiple versions of open-source AUTs [18]. The results show that PerfImpact performs much better in selecting inputs leading to performance regressions (i.e., 162% - 289% longer time differences between two versions as compare to random inputs) and is able to identify problematic code changes. Moreover, the results also show that those identified code changes have non-linearly increased execution times of their impact sets in the newly released version when workload is increasing. The major contribution of this work is in using MLs and GAs for selecting inputs for exposing bottlenecks via automating application profiling. The corresponding execution traces are further analyzed for locating problematic methods. We are also planning on tailoring our approaches to feature-level granularity [22, 23, 4, 5, 25, 3] in addition to method-level, like recovering traceability links between features and bottlenecks to detect risky features.

5. ACKNOWLEDGMENTS

This work is supported in part by the NSF CNS-1510239 and CCF-1253837 grants. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

6. REFERENCES

[1] T.-H. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora. Detecting performance anti-patterns for applications developed using object-relational mapping. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 1001–1012, 2014.

[2] W. W. Cohen. Fast effective rule induction. In *Twelfth ICML*, pages 115–123. Morgan Kaufmann, 1995.

[3] B. Dit, E. Moritz, and D. Poshyvanyk. A tracelab-based solution for creating, conducting, and sharing feature location experiments. In *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, pages 203–208, 2012.

[4] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95, 2013.

[5] B. Dit, M. Revelle, and D. Poshyvanyk. Integrating information retrieval, execution and link analysis algorithms to improve feature location in software. *Empirical Software Engineering*, 18(2):277–309, 2013.

[6] I. F. Emilio Coppa, Camil Demetrescu. Input-sensitive profiling. *TSE*, 40(12):1185–1205, 2014.

[7] S. Grant, J. R. Cordy, and D. Skillicorn. Automated concept location using independent component analysis. In *2008 15th Working Conference on Reverse Engineering*, pages 138–142, 2008.

[8] M. Grechanik, C. Fu, and Q. Xie. Automatically finding performance problems with feedback-directed learning software testing. In *ICSE'12*, pages 156–166, 2012.

[9] M. Grechanik, Q. Luo, D. Poshyvanyk, and A. Porter. Enhancing rules for cloud resource provisioning via learned software performance models. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ICPE '16, pages 209–214, 2016.

[10] 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*, ICSE 2014, pages 60–71, 2014.

[11] A. Hyvärinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Netw.*, 13(4-5):411–430, May 2000.

[12] 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*, PLDI '12, pages 77–88, 2012.

[13] J. Law and G. Rothermel. Whole program path-based dynamic impact analysis. In *ICSE '03*, pages 308–318.

[14] M. Linares-Vásquez, C. Vendome, Q. Luo, and D. Poshyvanyk. How developers detect and fix performance bottlenecks in android apps. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 352–361. IEEE, 2015.

[15] Q. Luo, K. Moran, and D. Poshyvanyk. A large-scale empirical comparison of static and dynamic test case prioritization techniques. In *Proceedings of The 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, 2016.

[16] Q. Luo, A. Nair, M. Grechanik, and D. Poshyvanyk. Forepost: A tool for detecting performance problems with feedback-driven learning software testing. *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 593–596, 2016.

[17] Q. Luo, A. Nair, M. Grechanik, and D. Poshyvanyk. Forepost: Finding performance problems automatically with feedback-directed learning software testing. *Empirical Software Engineering(EMSE)*, pages 1–51, 2016.

[18] Q. Luo, D. Poshyvanyk, and M. Grechanik. Mining performance regression inducing code changes in evolving software. In *Proceedings of the 13th International Workshop on Mining Software Repositories (MSR)*, pages 25–36. ACM, 2016.

[19] T. Nguyen, B. Adams, Z. M. Jiang, A. Hassan, M. Nasser, and P. Flora. Automated verification of load tests using control charts. In *APSEC '11*, pages 282–289.

[20] 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 *ICPE '12*, pages 299–310.

[21] S. Park, B. M. M. Hossain, I. Hussain, C. Csallner, M. Grechanik, K. Taneja, C. Fu, and Q. Xie. Carfast: Achieving higher statement coverage faster. In *FSE '12*, pages 35:1–35:11.

[22] D. Poshyvanyk, M. Gethers, and A. Marcus. Concept location using formal concept analysis and information retrieval. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21(4):23, 2012.

[23] D. Poshyvanyk and A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In *Proceedings of 15th IEEE International Conference on Program Comprehension(ICPC)*, pages 37–48. IEEE, 2007.

[24] M. Pradel, M. Huggler, and T. R. Gross. Performance regression testing of concurrent classes. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 13–25, New York, NY, USA, 2014. ACM.

[25] M. Revelle, B. Dit, and D. Poshyvanyk. Using data fusion and web mining to support feature location in software. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, pages 14–23, 2010.

[26] D. Shen, Q. Luo, D. Poshyvanyk, and M. Grechanik. Automating performance bottleneck detection using search-based application profiling. In *ISSTA'15*, pages 270–281.

[27] X. Xiao, S. Han, D. Zhang, and T. Xie. Context-sensitive delta inference for identifying workload-dependent performance bottlenecks. In *ISSTA '13*, pages 90–100, 2013.

[28] P. Zhang, S. Elbaum, and M. B. Dwyer. Automatic generation of load tests. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 43–52, 2011.