PerfoRT: A Tool for Software Performance Regression

Paulo Roberto Farah
prfarah@inf.ufpr.br
Federal University of Paraná (UFPR)
Curitiba, PR, Brazil

Silvia Regina Vergilio
silvia@inf.ufpr.br
Federal University of Paraná (UFPR)
Curitiba, PR, Brazil

ABSTRACT

In this paper, we present PerfoRT, a tool to ease software performance regression measurement of Java systems. Its main characteristics include: minimal configuration to ease automation and hide complexity to the end user; a broad scope of performance metrics including system, process, JVM, and tracing; and presentation of the results from a developer's perspective. We show some of its features in a usage example using Apache Commons BCEL project.

ACM Reference Format:
Paulo Roberto Farah and Silvia Regina Vergilio. 2023. PerfoRT: A Tool for Software Performance Regression. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3578245.3584928

1 INTRODUCTION

Performance regression testing is an approach to identify performance issues like response time degradation or resource utilization. It compares performance between successive versions of a software project using existing tests. Comparing performance between versions may prevent early-stage degradation and fix bugs in subsequent versions before they reach the users [2].

Some tools [3, 5] measure performance metrics and compare their values between versions, considering only one type of code element, such as class or method. Others [1, 4, 5] have as goal prioritization of performance regression tests to reduce the number of executed test cases. But overall, existing tools present at least one of the following limitations: i) the great majority does not associate performance metrics with executed code elements; ii) they implement a limited scope of performance metrics, in most cases only execution time and CPU; and iii) the process are partially automated. For example, the assessment steps are not supported and code needs to be added to measure the system under test.

In order to overcome these limitations, we introduce PerfoRT (**Perfo***rmance* **R***egression* **T***ool*), a tool that automates the measurement of performance regression in Java projects and helps developers to mine performance regressions of software repositories. PerfoRT allows developers to automatically extract runtime performance information from Java projects, such as the number of calls and time duration of versions, packages, classes, and methods. Moreover, it provides information related to testing code coverage metrics, process, and system utilization behavior, as well as to Java Virtual Machine (JVM) events. This demo paper describes PerfoRT architecture in Section 2, illustrates a use example in Section 3, and concludes in Section 4.

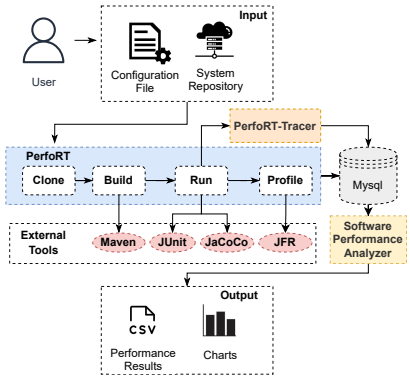

Figure 1: PerfoRT architecture.

2 PerfoRT ARCHITECTURE

PerfoRT profiles and traces different project versions using existing tests in the system repository. It encompasses three main modules depicted in Figure 1: PerfoRT-Core, PerfoRT-Tracer, and PerfoRT-Analizer[1]. PerfoRT requires Ubuntu operational system and Maven to run. The Java projects need to be compiled using JDK 8 or superior and installed by Maven without errors.

PerfoRT-Core is responsible for cloning and extracting properties of the target system such as authors, committers, files, and changes. Next, it identifies the used building tool, compiles, installs, and collects existing regression tests using Apache Maven. It instruments classes using JaCoCo Java code coverage library[2] to collect coverage metrics and executes tests using JUnit[3]. In summary, our tool collects system, process, and tracing metrics and saves the time of executed methods during runtime. The methods are filtered from the monitored package as specified in the settings file. Then, PerfoRT-Core retrieves information on test running processes and system utilization. It also combines JVM and test coverage metrics measured by the tools JFR[4] and JaCoCo, respectively.

PerfoRT profiles a broad set of performance aspects, including process, system, and virtual machine performance, execution time, CPU and memory usage, disk I/O, and network performance. The profiling and tracing measurement activities are fully automated processes and do not require developers to add code to the systems under test manually. The metrics monitored are chosen to identify performance issues related to finding bottlenecks, garbage

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0072-9/23/04.
https://doi.org/10.1145/3578245.3584928

[1]The source code is available at https://github.com/paulorfarah/perfort
[2]https://www.eclemma.org/jacoco/
[3]https://junit.org/junit5/
[4]https://docs.oracle.com/en/java/java-components/jdk-mission-control/8/user-guide/using-jdk-flight-recorder.html

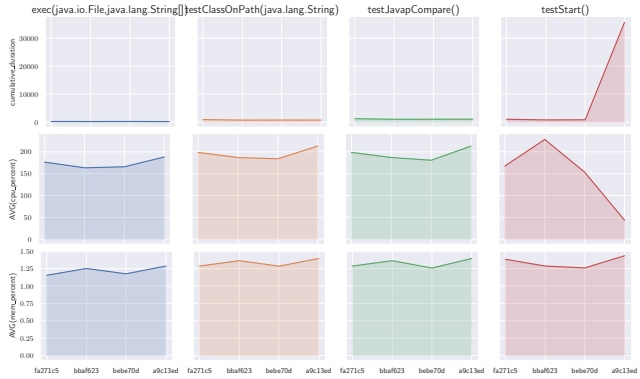

Figure 2: Test cases monitored from PLSETestCase.java.

collection, synchronization, I/O, code execution, memory usage, and testing coverage. Testing coverage can help in the identification of code parts that are being tested and of the most relevant test cases measured. A limitation of the tool is that all metrics are collected automatically and the users can not select metrics to measure.

Lastly, PerfoRT-Analyzer processes raw data and allows visualization of the results. It outputs performance regression-measured metrics to CSV files and charts. In this initial version of PerfoRT, this module is capable of: i) generating a statistical descriptive analysis of versions, a violin density chart by versions, including all methods of all collected classes and by a specific method; ii) generating a multiple area chart to evaluate the performance of all methods of a class; and iii) generating a waterfall chart of the execution time of methods called by test cases. As this module uses Python, users can create other analyses to better attend to their needs.

3 USAGE EXAMPLE

This section illustrates some of the inputs and outputs produced by PerfoRT through a usage example. To this end, we use the Apache Commons BCEL[5] byte code engineering library.

Figure 2 shows an example of the class scope. It contains the performance analysis of the test cases of the test file BCELifierTestCase.java of four releases identified by the hashes fa271c5, bbaf623, bebe70d, and a9c13ed. The file contains four methods: exec, testClassOnPath, testJavapCompare, and testStart. Three metrics are presented: cumulative duration of the execution time, and CPU and RAM memory percent usage. This output provides the developer with information about the set of test cases between evaluated versions. For example, we can observe that testStart() presented a significant increase in the cumulative duration for the release a9c13ed. On the other hand, it shows the smallest CPU usage of the evaluated releases.

At the method level, we can observe in Figure 3 an example of a waterfall chart of the cumulative duration of all methods called by test case testStart(), from test file BCELifierTestCase.java of release a9c13ed. It is created using the tracing metrics. The chart shows the time in seconds and the methods are in order from the bottom to the top. The testcase testStart() at the base of the figure calls BCELifierTestCase.getJavaClass(java.lang.String) and so on. As the chart is cumulative, bars at the bottom sum the time of all bars

[5]https://commons.apache.org/proper/commons-bcel/

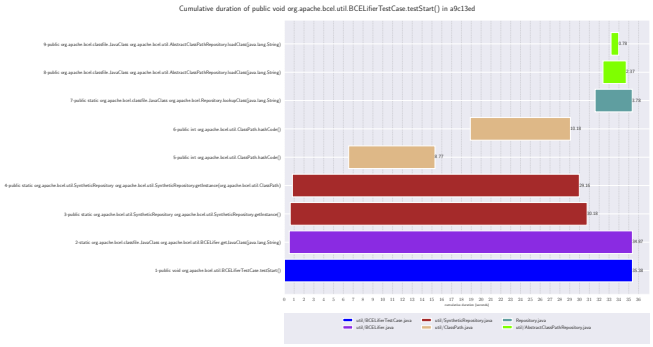

Figure 3: Execution time of methods called by the test case util.BCELifierTestCase.testStart().

above them. Each color of method call represents a different class. Thus, the developers can analyze and look for possible causes of a degradation in performance.

4 CONCLUDING REMARKS

This paper introduces PerfoRT, a tool to measure software performance using existing regression tests. The tool measures miscellaneous metrics of Java programs including method calls, execution time, testing code coverage, process and system utilization, and JVM events. In this way, PerfoRT allows developers, testers, and researchers to mine a robust set of software performance aspects of real-world projects in an automated fashion. It automates performance regression testing tasks since clone the project from GitHub to measure performance metrics and trace the target system and save results in a database. The results can help measure and analyze the performance of systems and create software performance detailed datasets for varied purposes.

PerfoRT is language-dependent (it works only for Java projects). The current functionalities PerfoRT-Analizer are also limited, but this module can be extended by the user according to their needs. Future work includes an automated generation of test cases to complement existing tests of the target systems.

ACKNOWLEDGMENTS

This work is supported by CNPq (Grant:305968/2018-1) and UDESC.

REFERENCES

[1] Jinfu Chen, Weiyi Shang, and Emad Shihab. 2022. PerfJIT: Test-Level Just-in-Time Prediction for Performance Regression Introducing Commits. *IEEE Transactions on Software Engineering* 48, 5 (2022), 1529–1544.
[2] Peng Huang, Xiao Ma, Dongcai Shen, and Yuanyuan Zhou. 2014. Performance Regression Testing Target Prioritization via Performance Risk Analysis. In *International Conference on Software Engineering (ICSE 2014)*. 60–71.
[3] Christoph Laaber and Philipp Leitner. 2017. (H|g)Opper: Performance History Mining and Analysis. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering (ICPE'17)*. ACM.
[4] Shaikh Mostafa, Xiaoyin Wang, and Tao Xie. 2017. PerfRanker: Prioritization of Performance Regression Tests for Collection-Intensive Software. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Santa Barbara, CA, USA) *(ISSTA 2017)*. ACM, New York, NY, USA, 23–34.
[5] David Georg Reichelt, Stefan Kühne, and Wilhelm Hasselbring. 2019. PeASS: A Tool for Identifying Performance Changes at Code Level. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1146–1149.