quiho: Automated Performance Regression Using Fine Granularity Resource Utilization Profiles

Ivo Jimenez
UC Santa Cruz
ivo.jimenez@ucsc.edu

Jay Lofstead
Sandia National Laboratories
gflofst@sandia.gov

Carlos Maltzahn
UC Santa Cruz
carlosm@ucsc.edu

ABSTRACT

We introduce *quiho*, a framework used in automated performance regression tests. *quiho* discovers hardware and system software resource utilization patterns that influence the performance of an application. It achieves this by applying sensitivity analiysis, in particular statistical regression analysis (SRA), using application-independent performance feature vectors to characterize the performance of machines. The result of the SRA, in particular feature importance, is used as a proxy to identify hardware and low-level system software behavior. The relative importance of these features serve as a performance profile of an application, which is used to automatically validate its performance behavior across revisions. We demonstrate that *quiho* can successfully identify performance regressions by showing its effectiveness in profiling application performance for synthetically induced regressions as well as several found in real-world applications.

1 INTRODUCTION

Quality assurance (QA) is an essential activity in the software engineering process [1–3]. Part of the QA pipeline involves the execution of performance regression tests, where the performance of the application is measured and contrasted against past versions [4–6]. Examples of metrics used in regression testing are throughput, latency, or resource utilization over time. These metrics are compared and when significant differences are found, this constitutes a regression.

One of the main challenges in performance regression testing is defining the criteria to decide whether a change in an application's performance behavior is significant, that is, whether a regression has occurred [7]. Simply comparing values (e.g. runtime) is not enough, even if this is done in statistical terms (e.g. mean runtime within a pre-defined variability range). Traditionally, this investigation is done by an analyst in charge of looking at changes, possibly investigating deeply into the issue and finally determining whether a regression exists.

Conference'17, July 2017, Washington, DC, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

When investigating a candidate of a regression, one important task is to find bottlenecks [8]. Understanding the effects in performance that distinct hardware and low-level system software[1] components have on applications is an essential part of performance engineering [9–11]. One common approach is to monitor an application's performance in order to understand which parts of the system an application is hammering on [5]. Automated solutions have been proposed [7,12–14]. The general approach of these is to analyze logs and/or metrics obtained as part of the execution of an application in order to automatically determine whether a regression has occurred. Most of them do this by creating prediction models that are checked against the runtime metrics. As with any prediction model, there is the risk of false/positive negatives **TODO: expand; put quiho in context**.

In this work, we present *quiho* an approach aimed at complementing automated performance regression testing by using system resource utilization profiles associated to an application. A resource utilization profile is obtained using Statistical Regression Analysis[2] (SRA) where application-independent performance feature vectors are used to characterize the performance of machines. The performance of an application is then analyzed applying SRA to build a model for predicting its performance, using the performance vectors as the independent variables and the application performance metric as the dependant variable. The results of the SRA for an application, in particular feature importance, is used as a proxy to characterize hardware and low-level system utilization behavior. The relative importance of these features serve as a performance profile of an application, which is used to automatically validate its performance behavior across multiple revisions of its code base.

In this article, we demonstrate that *quiho* can successfully identify performance regressions. We show (Section 5) that *quiho* (1) obtains resource utilization profiles for application that reflect what their codes do and (2) effectively uses these profiles to identify induced regressions as well as other regressions found in real-world applications. The contributions of our work are:

- A method for quantifying system performance via microbenchmark performance vectors.
- Insight: feature importance in SRA models (trained using these performance vectors) gives us a resource utilization profile of an application without having to look at the code.
- Methodology for evaluating automated performance regression. We introduce a set of synthetic benchmarks aimed at evaluating automated regression without the need of real repositories. These benchmarks take as input parameters

[1]Throughout this paper, we use "system" to refer to hardware, firmware and the operating system (OS).
[2]We use the term *Statistical Regression Analysis* (SRA) to differentiate between "traditional" regression analysis in software engineering and regression analysis in statistics.

that determine their performance behavior, thus simulating different "versions" of an application.
- A negative result: ineffectiveness of resource utilization profiles for predicting performance.

We give an overview SRA in Section **??** and its use in both, bottleneck identification as well as in performance regression testing. We then show the intuition behind *quiho* and how can be used to automate regression tests (Section **??**). We then do a more in-depth description of *quiho* (Section **??**), followed by our evaluation of this approach (Section 5). We briefly show how *quiho*'s resource utilization profiles can not be used to predict performance using some common machine learning techniques (Section 5.4). We then close with a brief discussion on challenges and opportunities enabled by *quiho* (Section 6).

2 STATISTICAL REGRESSION ANALYSIS IN SOFTWARE TESTING

SRA is an approach for modeling the relationship between variables, usually corresponding to observed data points [15]. One or more independent variables are used to obtain a *regression function* that explains the values taken by a dependent variable. A common approach is to assume a *linear predictor function* and estimate the unknown parameters of the modeled relationships.

2.1 Anomaly Detection and Bottleneck Identification

It's been used in bottleneck detection both [8]. **TODO: mention briefly how it is used**.

2.2 Automated Regression Testing

In [13], they use it to detect regressions using a dataset of performance counters.

3 MOTIVATION

TODO: show a diagram with an automated testing loop and show where quiho is placed

Traditionally, coarse-grained resource utilization (i.e. CPU-, memory- or IO-bound) can be obtained by monitoring an application's resource utilization over time. Fine granularity behavior allows application developers and performance engineers to quickly understand what they need to focus on while refactoring an application.

Fine granularity performance behavior, for example, system subcomponents such as the OS memory mapping submodule or the CPU's cryptographic unit are harder to find, and usually involve eyeballing source code, static code analysis, or analyzing hardware/OS performance counters.

An alternative is to infer a bottleneck by comparing the performance (e.g. runtime) of the same software stack (OS + application) on platforms with different hardware characteristics. For example, if we know that machine A has higher memory bandwidth than machine B, and an application is memory-bound, then this application will perform better on machine A. This approach presents three challenges:

1. Is difficult to obtain the performance characteristics of a machine by just looking at the hardware spec, so other more practical alternative is required. For example, the spec might specify that the machine has DDR4 memory sticks, with a theoretical peak throughput of XX GB/s, but the actual memory bandwidth is likely to be less than this.
2. We need to ensure that the software stack is the same on all machines where the application runs.
3. The amount of effort required to run applications on a multitude of platforms is not negligible.

4 OUR APPROACH

In this section we describe:

- Performance vectors to characterize behavior
- Using these to build SRA models for application performance.
- Feature importance in SRA models gives us resource utilization profiles.

4.1 Performance Feature Vectors As System Performance Characterization

While the hardware and software specification can serve to describe the performance characteristics of a machine, the real performance characteristics can only feasibly[3] be obtained by executing programs and capturing metrics. The question then boils down to which programs should we use to characterize performance? Ideally, we would like to have many programs that execute every possible opcode mix so that we measure their performance. Since this is an impractical solution, an alternative is to create synthetic microbenchmarks that get as close as possible to exercising all the available features of a system.

`stress-ng` is a tool that is used to "stress test a computer system in various selectable ways. It was designed to exercise various physical subsystems of a computer as well as the various operating system kernel interfaces". There are multiple stressors for CPU, CPU cache, memory, OS, network and filesystem. Since we focus on system performance bandwidth, we execute the (as of version 0.07.29) 42 stressors for CPU, memory and virtual virtual memory stressors. A "stressor" is a routine that loops a function multiple times and reports the rate of iterations executed for a determined period of time (referred to as `bogo-ops-per-second`).

Using this battery of stressors, one can obtain a performance profile of a machine. When this profile is compared against the profile of another machine, we can quantify the difference in performance between the two. Fig. 1 shows the correlation matrix of stressors for all the distinct machine configurations available in CloudLab [16] (shows a summary of hardware specs). Every stressor can be mapped to basic features of the underlying platform. For example, `stream` to memory bandwidth, `zero` to memory mapping, `qsort` to sorting data, etc.

4.2 System Resource Utilization Via Feature Importance in Regression Models

The performance of an application is determined by the performance of the subcomponents that get stressed the most by the

[3]One can get generate arbitrary performance characteristics by interposing a hardware emulation layer and deterministically associate performance characteristics to each instruction based on specific hardware specs. While possible, this is impractical (we are interested in characterizing "real" performance).

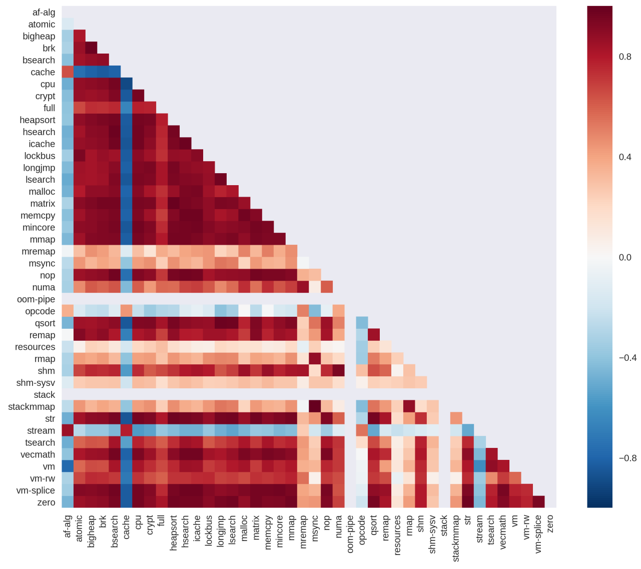

Figure 1: Correlation matrix of performance vectors.

application’s code. Thus, intuitively, if the performance of an application across multiple machines resembles the . In Section 5 we empirically validate this. In this section we explain the regression technique used.

Linear regression:

- generic regression
- gradient boosting trees
- normalization

5 EVALUATION

This section we demonstrate:

- resource utilization profiles (RUP) accurately capture application performance behavior.
- RUPs work for synthetic workloads.
- RUPs work for “real” regressions.

Popper: This paper adheres to The Popper Convention[4] [17], so experiments presented here are available in the repository for this article[5]. Experiments can be examined in more detail, or even re-run, by visiting the `[source]` link next to each figure. That link points to a Jupyter notebook that shows the analysis and source code for that graph, which points to an experiment and its artifacts.

5.1 Performance Profiles

We show they actually show the performance of known benchmarks.

Methodology - For every workload:

1. Discover relevant features using quiho.
2. Analyze code to corroborate that discovered features are indeed the cause of performance differences.

[4]http://falsifiable.us
[5]http://github.com/ivotron/quiho-popper

5.2 Simulating Regressions

We show that if we simulate regressions, then *quiho* identifies them correctly.

5.3 Real-world Scenarios

We show that *quiho* works with real regressions. Systems:

- zlog
- mariadb
- apache commons math
- gcc

5.4 *quiho* cannot predict performance

We show how *quiho* does not do a good job at predicting performance.

6 CONCLUSION AND FUTURE WORK

In the not-so-distant future:

- multi-node
- minimum number of machines?
- single machine?
- long-running (multi-stage) applications. e.g. a web-service or big-data application with multiple stages. In this case, we would define windows of time and we would apply quiho to each. The challenge: how do we automatically get the windows rightly placed.

Acknowledgments: This work was partially funded by the Center for Research in Open Source Software[6], Sandia National Laboratories and NSF Awards #1450488.

7 REFERENCES

[1] G.J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 2011.

[2] A. Bertolino, “Software Testing Research: Achievements, Challenges, Dreams,” *2007 Future of Software Engineering*, 2007.

[3] B. Beizer, *Software Testing Techniques*, 1990.

[4] J. Dean and L.A. Barroso, “The tail at scale,” *Commun ACM*, vol. 56, Feb. 2013.

[5] B. Gregg, *Systems Performance: Enterprise and the Cloud*, 2013.

[6] F.I. Vokolos and E.J. Weyuker, “Performance Testing of Software Systems,” *Proceedings of the 1st International Workshop on Software and Performance*, 1998.

[7] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, “Anomaly? Application change? Or workload change? Towards automated detection of application performance anomaly and

[6]http://cross.ucsc.edu

change," *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, 2008.

[8] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance Anomaly Detection and Bottleneck Identification," *ACM Comput Surv*, vol. 48, Jul. 2015.

[9] G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu, "Understanding and Detecting Real-world Performance Bugs," *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2012.

[10] S. Han, Y. Dang, S. Ge, D. Zhang, and T. Xie, "Performance Debugging in the Large via Mining Millions of Stack Traces," *Proceedings of the 34th International Conference on Software Engineering*, 2012.

[11] M. Jovic, A. Adamoli, and M. Hauswirth, "Catch Me if You Can: Performance Bug Detection in the Wild," *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, 2011.

[12] Z.M. Jiang, "Automated Analysis of Load Testing Results," *Proceedings of the 19th International Symposium on Software Testing and Analysis*, 2010.

[13] W. Shang, A.E. Hassan, M. Nasser, and P. Flora, "Automated Detection of Performance Regressions Using Regression Models on Clustered Performance Counters," *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, 2015.

[14] C. Heger, J. Happe, and R. Farahbod, "Automated Root Cause Isolation of Performance Regressions During Software Development," *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 2013.

[15] D.A. Freedman, *Statistical Models: Theory and Practice*, 2009.

[16] R. Ricci and E. Eide, "Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications,"*;login:* vol. 39, 2014/December.

[17] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "The Popper Convention: Making Reproducible Systems Evaluation Practical," *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017.