quiho: Automated Performance Regression Testing Using Inferred Resource Utilization Profiles

Mariette Souppe
UC Santa Cruz
msouppe@ucsc.edu

Kerry Veenstra
UC Santa Cruz
veenstra@ucsc.edu

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

Katia Obraczka
UC Santa Cruz
katia@soe.ucsc.edu

ABSTRACT

In the mobile and wireless network domain there are a lot of simulations and domain tools that are required to be produce results of an experiment. Our approach is to minimize the amount of time spent to set up experiments and assumptions for experiments and be able to reproduce experiments with available tools for any user to conduct an experiment on their own environment with no clashing dependencies. Using Popper, a convention and CLI tool, and Docker, a container image, allows any user to reproduce experiments from their peers or a research paper with the exact parameters and assumptions has the author made. This allows for easy replication and reproducibility.

CCS CONCEPTS

• **Software and its engineering** → **Software performance**; **Software testing and debugging**; **Acceptance testing**; *Empirical software validation*; • **Social and professional topics** → *Automation*;

ACM Reference Format:
Mariette Souppe, Kerry Veenstra, Ivo Jimenez, and Katia Obraczka. 2018. *quiho*: Automated Performance Regression Testing Using Inferred Resource Utilization Profiles. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering, April 9–13, 2018, Berlin, Germany.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3184407.3184422

1 INTRODUCTION

2 METHODOLOGY

2.1 Popper

2.2 Docker

In order to achieve a reproducibility model for experiments so that an experiment can truly run on any personal machine the technology tool called Docker is used to accomplish this goal. Docker, a technology container, creates an environment which packages an application with all of the application's dependencies. In the experiment, which is further explained [in section], this tool is used the experiment needs Java and Python installed on a machine in order for the whole experiment to run properly. Normally, one would have to make sure that both of these dependencies are installed on one's personal machine, however with the use of Docker an environment is created with those dependencies

This enables the portability of an experiment which further helps achieve reproducibility.

ICPE '18, April 9–13, 2018, Berlin, Germany
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5095-2/18/04…$15.00
https://doi.org/10.1145/3184407.3184422

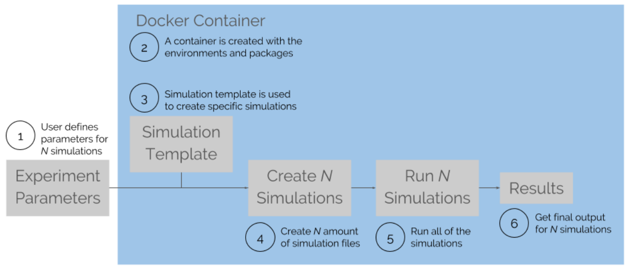
Docker Container
2 A container is created with the environments and packages
3 Simulation template is used to create specific simulations
1 User defines parameters for *N* simulations
Simulation Template
Experiment Parameters
Create *N* Simulations
Run *N* Simulations
Results
4 Create *N* amount of simulation files
5 Run all of the simulations
6 Get final output for *N* simulations

Figure 1: figure caption goes in here

2.3 Contiki

2.4 Cooja

For the experiment, to be described in the next section, uses Cooja to conduct the main experiment. Cooja is a network simulator that is used in wireless sensor networks which allows to simulate small or large networks.

3 EXAMPLE

3.1 Background

3.2 Pipeline

Figure X shows the pipeline for the this experiment and all of the

- **.popper.yaml**
  This file consists the ordering in which popper executes the main scripts; setup.sh, run.sh, post-run.sh to run the entire experiment with out extra input.
- **.git**
  Currently the full experiment is not on Git, which it will be soon, however having the whole pipeline on Git will achieve the portable and reproducibility goal. Any user will be able to clone the repository and be able to reproduce the experiment with out having to individually download dependecies.

- **Dockerfile**
  Retrieves the docker image with Java and installs Python, pip, java-random, pyyaml, and jinja2
- **setup.sh**
  Builds a docker container with the environmnents, dependencies, and packages that are needed to run the pipeline.
- **run.sh**
  Will run the simulations in the environment that has been previously defined.
- **sim_config.yaml**
  The simulation configration file allows the user to run multiple experiments with different values for the parameters. The parameters that the user can define are the filename, decide to use a random generator true/false for nodes placement, random seed, and number of nodes (maximum of 50). The filename is used to name the different experiments so when looking at the final output it will be evident which experiment produce certain results. As of right now the algorithm has hard coded values for the initial placement of the nodes, but as the experiment is further developed the random generator will radomly generator the node placement. [Add in about random seed and point of that] The user can also define the amount of nodes desired for the experiment with a maximum of 50 nodes, since that is the maximum the algorithm of the experiment can handle.
- **sim_template.csc**
  The template file consists of how the simulator reads in script to create the simulations based on the defined parameters for each experiment.
- **create_sim_files.py**
  For the N amount of simulations defined in the sim_config.yaml, this file will merge the parameters with the template to create N amount of simlations.
- **simulations directory**
  All of the simulation files that were produced from create_sim_files.py which are then run in run.sh.
- **output directory**
  The output for each of the simulations containing the final visability value between the nodes [Need better wording and explain that a bit more]
- **contiki/tools/cooja**
  The main experiment files are located, where Cooja is the simulator used for the experiment.

Correlation-based Analysis and Supervised Learning. Correlation and supervised learning approaches have been proposed in the context of software testing, mainly for detecting anomalies in application performance [1]. In the former, runtime performance metrics are correlated to application performance using a variety of distinct metrics. In supervised learning, the goal is the same (build prediction models) but using labeled datasets. Decision trees are a form of supervised learning, however, given that *quiho* applies regression rather than classification techniques, it does not rely on labeled datasets. Lastly, *quiho* is not intended to be used as a way of detecting anomalies, although we have not analyzed its potential use in this scenario.

4 FUTURE WORK

The main limitation in *quiho* is the requirement of having to execute a test on more than one machine in order to obtain IRUPs. On the other hand, we can avoid having to run `stress-ng` every time the application gets tested by integrating this into the infrastructure (e.g., system administrators can run `stress-ng` once a day or once a week and make this information for every machine available to users).

We are currently working in adapting this approach to profile distributed and multi-tiered applications. We also plan to analyze the viability of applying *quiho* in multi-tenant configurations and to profile long-running (multi-stage) applications such as a web-service or big-data applications. In these cases, we would define windows of time and apply *quiho* to each. The main challenge in this scenario is to automatically define the windows in such a way that we can get accurate profiles.

In the era of cloud computing, even the most basic computer systems are complex multi-layered pieces of software, whose performance properties are difficult to comprehend. Having complete understanding of the performance behavior of an application, considering the parameter space (workloads, multi-tenancy, etc.) is challenging. One application of *quiho* we have in mind is to couple it with automated black-box (or even gray-box) testing frameworks to improve our understanding of complex systems.

Acknowledgments: We would like to thank Bernardo Gonzalez for his feedback on a preliminary version of this paper, as well as all the anonymous reviewers. Special thanks go to our ICPE shepherd. This work was partially funded by the Center for Research in Open Source Software[1], Sandia National Laboratories, NSF Award #1450488 and DOE Award #DE-SC0016074. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

REFERENCES

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

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