

Demo Abstract: PopperCI: Automated Reproducibility Validation

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

Sina Hamedian
UC Santa Cruz
sina@ucsc.edu

Jay Lofstead
Sandia National Laboratories
gflofst@sandia.gov

Carlos Maltzahn
UC Santa Cruz
carlosm@cs.ucsc.edu

Kathryn Mohror
Lawrence Livermore National Laboratory
kathryn@llnl.gov

Remzi Arpaci-Dusseau
UW Madison
remzi@cs.wisc.edu

Andrea Arpaci-Dusseau
UW Madison
dusseau@cs.wisc.edu

Robert Ricci
University of Utah
ricci@cs.utah.edu

In this demo we illustrate the usage of PopperCI [1], a continuous integration (CI) service for experiments hosted at UC Santa Cruz that allows researchers to automate the end-to-end execution and validation of experiments. PopperCI assumes that experiments follow Popper [2], a convention for implementing experiments and writing articles following a DevOps approach that has been proposed recently.

I. POPPER, EXPERIMENT VALIDATIONS AND POPPERCI

A. Popper

Popper [2] revisits the idea of an executable paper, which proposes the integration of executables and data with scholarly articles to help facilitate its reproducibility. The goal of Popper is to implement executable papers in today's cloud-computing world by treating an article as an open source software (OSS) project. Popper is realized in the form of a convention for systematically implementing the different stages of the experimentation process following a DevOps [3] approach. The convention can be summarized in three high-level guidelines:

1. Pick a DevOps tool for each stage of the scientific experimentation workflow.
2. Put all associated scripts (experiment and manuscript) in version control, in order to provide a self-contained repository.
3. Document changes as experiment evolves, in the form of version control commits.

By following these guidelines researchers can make all associated artifacts publicly available with the goal of minimizing the effort for others to re-execute and validate experiments.

B. Experiment Validations

One optional but important component in Popper is the validation of experiments by explicitly codifying expectations. These domain-specific tests ensure that the claims made about the results of an experiment are valid after every re-execution. An example of this is performance regression testing done in software projects (e.g. [ScalaMeter](#)). In general, this can be part of the analysis/visualization phase of the experimentation workflow. To illustrate this stage further, consider an experiment

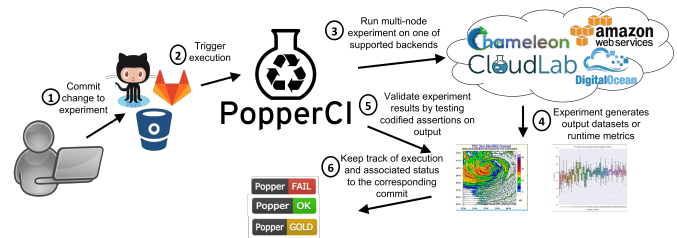


Figure 1: The continuous integration (CI) PopperCI service automates the execution and validation of experiments that run on public, private or government-funded cloud infrastructures. The status of an experiment execution is reported at <http://ci.falsifiable.us>.

that measures the scalability of the system as the number of nodes increases. One assertion might be like the following:

Listing 1 Example validation in the Aver language.

```
WHEN
  NOT network_saturated AND num_nodes=*
EXPECT
  system_throughput >= (baseline_throughput * 0.9)
```

The above is written in the Aver¹ [4] language and expresses linear scalability with respect to the underlying raw performance, i.e. “regardless of the number of nodes in the system, its throughput is always at least 90% of the raw performance”. The boolean value for `network_saturated` comes from network metrics that are captured at runtime. For example, some switches implement the SNMP protocol that allows to identify if the network is getting saturated. In general, for experiments in the computer and networking systems research domain, most of the data that is used at this stage comes from capturing runtime metrics about the underlying resources. Monitoring tools such as [Nagios](#) and [collectd](#) can be used

¹Aver is a language and tool that can be used to check the integrity of runtime performance metrics that claims make reference to. The tool evaluates simple if-then statements in SQL-like syntax against metrics captured in tabular format files (e.g. CSV files).

for this purpose.

C. PopperCI

Following the Popper convention results in producing self-contained experiments and articles, and reduces significantly the amount of work that a reviewer or reader has to undergo in order to re-execute experiments. However, it still requires manual effort in order to re-execute an experiment. The idea behind PopperCI is simple: by structuring a project in a commonly agreed way, experiment execution and validation can be automated without the need for manual intervention. Experimenters that follow this structure can make use of [PopperCI](#), a continuous integration (CI) service hosted at UC Santa Cruz that allows researchers to automate the end-to-end execution and validation of experiments. PopperCI runs experiments on public, private or government-funded cloud infrastructures in a fully automated way. In addition to this, the status of an experiment (integrity over time) can be tracked by the service hosted at <http://ci.falsifiable.us>.

II. DEMO

The goal of this demo is to illustrate the usefulness of PopperCI and the ease with which users trigger experiment executions. The audience will be guided through a practical example of how to implement a Popper-compliant experiment and how they can use PopperCI. We will walk them through the steps that are required to implement an experiment following the Popper convention, and how they can trigger experiments on multiple clouds by using PopperCI (we will show a step-by-step execution of the diagram in Fig. 1).

This demo consists of a multi-node experiment being seamlessly executed on two clouds (DigitalOcean and CloudLab). The experiment is a distributed machine learning (ML) benchmark on Spark. Due to space limitations we leave out the details of the setup but refer the reader to the Popper repository for this paper at <https://github.com/ivotron/popperci-paper>. The main goal of the experiment is to test a simple hypothesis: doubling the number of cores that each Spark worker has available to it should reduce the amount of time it takes to execute the ML benchmark.

This experiment makes use of Docker for packaging the software stack, Ansible to orchestrate the logic of the experiment, Aver to verify the output of the experiment and Terraform/GENI to specify the resources needed to execute an experiment. When a new commit is pushed to the main branch of this paper repository, a git hook registered at GitHub triggers the execution of the experiment at PopperCI. Before the experiment executes, basic checks on the experiment artifacts are executed: Docker images are built, Ansible scripts' syntax is checked and Terraform/GENI configuration files are sanitized.

Terraform's DSL allows to succinctly specify the resources that an experiment needs. In this case, we request 6 machines with the same characteristics. An example of how this is specified is shown in Lst. 2. We have as many of these node specifications as nodes in the experiment (6 in this case). The

resource request for CloudLab uses the GENI API and looks a bit similar (Lst. 3).

Listing 2 Terraform configuration for requesting a Droplet.

```
resource "digitalocean_droplet" "web" {
  image = "docker-ubuntu-16-04-x64"
  name   = "node1"
  region = "sf2"
  size   = "16gb"
}
```

Listing 3 CloudLab script for requesting a node.

```
import geni.portal as portal
import geni.rspec.pg as rspec

request = portal.context.makeRequestRSpec()
node1 = request.RawPC("node1")
node1.disk_image = "urn:publicid:IDN+image/UBUNTU16-64-STD"
portal.context.printRequestRSpec()
```

The output dataset for this experiment is in tabular format with two columns `num_workers` and `runtime`. The condition that is checked for this experiment is the following:

Listing 4 Validation for this use case.

```
EXPECT
runtime(num_workers=8) < runtime(num_workers=4)
```

We note that while the performance numbers obtained are relevant, they are not our main focus. Instead, we put more emphasis on how we can verify domain-specific validations, how we can reproduce results on multiple environments with minimal effort, and how we can ensure the validity of the results. This demo will show how PopperCI allows experimenters to execute on multiple clouds and corroborate claims about their systems on distinct platforms.

III. BIBLIOGRAPHY

- [1] I. Jimenez, S. Hamedian, J. Lofstead, C. Maltzahn, K. Mohror, R. Arpaci-Dusseau, A. Arpaci-Dusseau, and R. Ricci, "PopperCI: Automated Reproducibility Validation," *Proceedings of CNERT '17*, 2017.
- [2] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Standing on the Shoulders of Giants by Managing Scientific Experiments Like Software," *USENIX; login*, vol. 41, Nov. 2016.
- [3] G. Kim, J. Humble, P. Debois, and J. Willis, *The DevOps Handbook*, O'Reilly Media, 2016.
- [4] I. Jimenez, C. Maltzahn, J. Lofstead, A. Moody, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "I Aver: Providing Declarative Experiment Specifications Facilitates the Evaluation of Computer Systems Research," *TinyToCS*, vol. 4, 2016.