Practical Reproducible Evaluation of Computer Systems

Ivo Jimenez, Michael Sevilla, Noah Watkins, Sina Hamedian, Pete Wilcox, Carlos Maltzahn, Jay Lofstead, Kathryn Mohror, Adam Moody, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau
Problem of Reproducibility in Computation and Data Exploration

- What compiler was used?
- Which compilation flags?
- How was subsystem X configured?
- How does the workload look like?
- What parameters can be modified?
- What if I use input dataset Y?
- And if I run on platform Z?
- …
Lab Notebook

Results of running base-vs-targets for stressing on 4 machines

Results of running base-vs-targets for stressing on 4 machines

Base machine is "iissm-12" and targets were turned with the 'crafty' and 'c-nry' benchmarks.

The main difference between these results and the ones that appear on our blog is that we are reflecting the speedup function for w.d., for both (1) turning targets and (2) displaying results. This allows us to show better the reduction in variability without having to deal with different scales (slopes that lie between the (R-1) range are instead reflected and treated as speedups).

In short, we now unambiguously observe reduced variability when targets are limited. A couple of outliers, in particular stressing a more stressful, when limited, is very slow on target machines.

"..."
End-to-end Scientific Experimentation Pipelines

- observation or simulation
- data cleaning and processing
- data analysis
- output data and visualization

Manage Code
Data Mgmnt
Analyze/Visualize
Build and Package
Execute
Manuscript
Analogies With Modern SE Practices (aka DevOps)

<table>
<thead>
<tr>
<th>Scientific exploration</th>
<th>Software project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment code</td>
<td>Source code</td>
</tr>
<tr>
<td>Data management</td>
<td>Test examples</td>
</tr>
<tr>
<td>Analysis / visualization</td>
<td>Test analysis</td>
</tr>
<tr>
<td>Validation</td>
<td>CI / Regression testing</td>
</tr>
<tr>
<td>Manuscript / notebook</td>
<td>Documentation / reports</td>
</tr>
</tbody>
</table>

**Key Idea:** manage a scientific exploration like software projects
SciOps

DevOps View of The Experimentation Pipeline

observation or simulation → data cleaning and processing → data analysis → output data and visualization

Manage Code

Build and Package

Data Mgmt

Execute

Analyze/Visualize

output data and visualization
What is DevOps?

Typical

DevOps

$ bash myscript.sh
The Popper Convention

1. Pick one or more tools from the DevOps toolkit
2. Write scripts for an experiment pipeline
3. Put all scripts in a version control repository

Popper CLI tool

• Make it **super easy** to automate execution and validation of experimentation pipelines
  • easy → low-overhead → more likely it’ll be used

• Common convention to organize the contents of a repo

• CLI tool that helps users to implement pipeline stages

• Provide domain-specific examples
  • Today: Genomics, MPI, Ceph, Atmospheric Science
  • Working with domain-experts to contribute more examples
Common convention
to organize the
contents of a repo
$ cd my-paper-repo
$ git init
Initialized empty Git repository in my-paper-repo/.git
$ popper init
Initialized popper repository.
$ popper pipeline init mypipeline --stages=prepare,execute,cleanup
-- Initialized exp1 pipeline.

$ ls -l pipelines/mypipeline
total 20K
-rw-r----- 1 ivo ivo  8 Apr 29 23:58 README.md
-rw-r-x--- 1 ivo ivo 210 Apr 29 23:58 execute.sh
-rw-r-x--- 1 ivo ivo  206 Apr 29 23:58 prepare.sh
-rw-r-x--- 1 ivo ivo  61 Apr 29 23:58 cleanup.sh

#!/bin/bash
# request remote resources
docker run google/cloud-sdk
...

#!/bin/bash
# trigger execution of experiment
docker run google/kubectl run ...
...
```
provider:
type: chameleonbaremetal
image_name: CC-Ubuntu14.04-Docker
subnet: {name: shared-subnet1}
walltime: '00:30:00'
resources:
  storage:
    compute: 10
docker run --rm \\
  -e OS_AUTH_URL=$OS_AUTH_URL \\
  -e OS_TENANT_ID=$OS_TENANT_ID \\
  -e OS_TENANT_NAME=$OS_TENANT_NAME \\
  -e OS_PROJECT_NAME=$OS_PROJECT_NAME \\
  -e OS_USERNAME=$OS_USERNAME \\
  -e OS_PASSWORD=$OS_PASSWORD \\
  -e OS_KEYNAME=$OS_KEYNAME \\
  -v `pwd`/enos:/enos \\
  --workdir=/enos \\
  ivotron/enos:2.3.0
```
$ popper run exp1

Popper run started

Stage: setup.sh ..... 
Stage: run.sh ............... 
**Stage: validate.sh .** 
Stage: teardown.sh ...

Popper run finished 

Status: OK
Codified Validations

- Log file
- CSV
- DB Table
- TSDB
- ...

expect linear(num_nodes, throughput)

when not net_saturated
expect throughput >= (raw_bw * 0.9)

Stage: run.sh ............
Stage: validate.sh ....

[true] check linear scalability
[true] check system throughput

Popper run finished
Status: GOLD

[1]: Jimenez et al. Tackling the reproducibility problem in storage systems research with declarative experiment specifications, PDSW '15.
In software engineering, continuous integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day.

source: https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html
## ACM/Popper Badges

<table>
<thead>
<tr>
<th>Result Status</th>
<th>Artifacts</th>
<th>Re-executed By</th>
<th>ACM</th>
<th>Popper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeatability</td>
<td>Original</td>
<td>Original Author(s)</td>
<td></td>
<td>Popper GOLD</td>
</tr>
<tr>
<td>Replicability</td>
<td>Original</td>
<td>3rd Party</td>
<td>Popper GOOD</td>
<td>Popper GOLD</td>
</tr>
<tr>
<td>Reproducibility</td>
<td>Re-implemented</td>
<td>Anyone</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Archiving/DOI service integration

$ popper archive --zenodo --user=ivotron --password=****
Creating archive for repository on Zenodo.
|############################################################| 100 %

Your DOI link is: https://zenodo.org/record/1165550
Push-button Reproducible Evaluation ⇒ New Possibilities and Challenges

New:
• SciOps approach. End-to-end automated execution & validation.
• Improve the study of computer systems. Portability allows to fix the SW stack; we can now easily report results w.r.t. distinct environments.

Challenges:
• Larger search space: entire software stack can be parameterized.
• Cannot execute on new platforms (hardware does not exist yet).
• Identifying root causes of irreproducibility.