Self-verifiable Experimentation Pipelines: An Alternative to Research Artifact Descriptions

Ivo Jimenez and Carlos Maltzahn (*UC Santa Cruz*)

We make the case for incorporating self-verifiable experimentation pipelines in journals and conferences as a way of streamlining the review process for code and data associated to academic articles.

1 Introduction

Reproducibility is the cornerstone of the scientific method. Yet, in computational and data science domains, a gap exists between current practices and the ideal of having every new scientific discovery be *easily* reproducible [1]. Advances in computer science (CS) and software engineering slowly and painfully make their way into these domains—even in CS research itself [2,3], paradoxically.

To address some of the reproducibility issues we face today, a growing number of Computer Science conferences and journals incorporate an artifact evaluation process in which authors of articles submit artifact descriptions that are tested by a committee, in order to verify that experiments presented in a paper can be re-executed by others. In short, an artifact description is a 2-3 page narrative on how to replicate results, including steps that detail how to install software and how to re-execute experiments and analysis contained in a paper.

2 Self-verifiable Pipelines

An alternative to the manual creation and verification of an Artifact Description (AD) is to use a continuous integration (CI) service[1] such as Jenkins[2] to ensure that the code (and data) associated to an article can be re-executed by others. If the execution and validation of an experimentation pipeline using a CI service, the URL pointing to the project on the CI server that holds execution logs, as well as the repository containing all the automation scripts, can potentially be used as an alternative to ADs. In other words, the repository containing the code for experimentation pipelines, and the associated CI project,

[1] https://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software

[2] https://jenkins.io

serve both as a *Self-verifiable Experimentation Pipeline* (SEP). Thus, instead of submitting manually written ADs, authors can submit a link to the code repository where scripts for their pipelines reside, along with a link to the CI server that executes them, and use this as the basis for evaluating the reproducibility of their work.

While automating the execution of a pipeline can be done in many ways, in order for this approach to serve as an alternative to ADs, there are five high-level tasks that SEPs must carry out in every execution:

1. **Code and data dependencies**. Code must reside on a version control system (e.g. Github[3], Gitlab[4], etc.). If datasets are used, then they should reside in a dataset management system (datapackage, gitlfs, dataverse, etc.). The experimentation pipelines must obtain the code/data from these services on every execution.
2. **Setup**. The pipeline should build and deploy the code under test. For example, if a pipeline is using containers or VMs to package their code, the pipeline should build the container/VM images prior to executing them. The goal of this is to verify that all the code and 3rd party dependencies are available at the time a pipeline runs, and that software can be build correctly.
3. **Resource allocation**. If a pipeline requires a cluster or custom hardware to reproduce results, resource allocation must be done as part of the execution of the pipeline. This allocation can be static or dynamic. For example, if an experiment runs on custom hardware, the pipeline can statically allocate (i.e. hardcode IP/hostnames) the machines where the code under study runs (e.g. GPU/FPGA nodes). Alternatively, a pipeline can dynamically allocate nodes on CloudLab [4], Grid500K [5], Chameleon [6], or a public cloud provider. These services typically offer infrastructure automation tools such as Geni-lib[5] (Cloudlab), Enos [7] (Chameleon), SLURM [8] (HPC centers), or Terraform[6] (AWS, GCP, etc.). All these tools can be used to automate infrastructure-related tasks.
4. **Environment capture**. Capture information about the runtime environment. For example, hardware description, OS, system packages (i.e. software installed by system administrators), remote services (e.g. a scheduler). Many open-source tools can aid in aggregating this information such as SOSReport[7] or Facter[8].
5. **Validation**. Scripts must verify that the output corroborates the claims made on the article. For example, the pipeline might check that the throughput of a system is within an expected confidence interval (e.g. defined with respect to a baseline obtained at runtime), or that a numerical computation is within some expected bounds.

A list of Popper[9] [9] pipelines meeting the

[3]https://github.com
[4]https://gitlab.com
[5]https://bitbucket.org/barnstorm/geni-lib
[6]https://terraform.io
[7]https://github.com/sosreport/sos
[8]https://github.com/puppetlabs/facter
[9]A SEP is, by the way is defined in Section 2,

above criteria are available at Popper's official documentation website[10].

3 Review Process

We now illustrate how the peer-review process might work for a conference or journal that incorporates SEPs as a way of having authors demonstrate the reproducibility of their results. The process might look like this:

1. Authors create SEPs as part of their work.
2. Authors submit link(s) to the CI service that automatically executes the experimentation pipeline(s) associated to their submission.
3. Reviewers check logs of CIs to verify that the 5 components described in Section 2 are met.

Thus, incorporating SEPs to the peer-review process has two important properties:

- Improved Experimentation Practices. Authors that consider reproducibility as a first-class goal of their work increase the confidence in their results.
- Low-overhead Evaluations. With SEPs, the burden of ensuring that a pipeline can be re-executed by others relies on the authors, wich leverage the CI service as that third-person ensuring the help of CI),

a set of high-level requirements, rather than a detailed list of requisites. Consequentially, there are many alternatives to create a SEP. While the examples we link above are of Popper pipelines, this article has the intention to apply, in general, to any type of automated approach.

[10] https://popper.rtfd.io/en/latest/sections/examples.html.

References

[1] D.L. Donoho, A. Maleki, I.U. Rahman, M. Shahram, and V. Stodden, "Reproducible Research in Computational Harmonic Analysis," *Computing in Science & Engineering*, vol. 11, Jan. 2009.

[2] G. Fursin, "Collective Mind: Cleaning up the research and experimentation mess in computer engineering using crowdsourcing, big data and machine learning," Aug. 2013. Available at: http://arxiv.org/abs/1308.2410.

[3] C. Collberg and T.A. Proebsting, "Repeatability in Computer Systems Research," *Communications of the ACM*, vol. 59, Feb. 2016.

[4] R. Ricci and E. Eide, "Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications,"*;login:* vol. 39. Available at: http://www.usenix.org/publications/login/dec14/ricci.

[5] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche, "Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed," *Int. J. High Perform. Comput. Appl.*, vol. 20, Nov. 2006.

[6] J. Mambretti, J. Chen, and F. Yeh, "Next Generation Clouds, the Chameleon Cloud Testbed, and Software Defined Networking (SDN)," *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, 2015.

[7] R. Cherrueau, D. Pertin, A. Simonet, A. Lebre, and M. Simonin, "Toward a Holistic Framework for Conducting Scientific Evaluations of OpenStack," *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017.

[8] A.B. Yoo, M.A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, eds., 2003. Available at: http://link.springer.com/chapter/10.1007/10968987_3.

[9] 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.