# The Popper Convention: Making Reproducible Systems Evaluation Practical

Ivo Jimenez, Michael Sevilla, Noah Watkins, Carlos Maltzahn
*UC Santa Cruz*
<ivo,msevilla,jayhawk,carlosm>@cs.ucsc.edu

Jay Lofstead
*Sandia National Laboratories*
gflofst@sandia.gov

Kathryn Mohror
*Lawrence Livermore National Laboratory*
kathryn@llnl.gov

Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau
*UW Madison*
<dusseau,remzi>@cs.wisc.edu

***Abstract*—Independent validation of experimental results in the field of systems research is a challenging task, mainly due to differences in software and hardware in computational environments. Recreating an environment that resembles the original is difficult and time-consuming. In this paper we introduce *Popper*, a convention based on a set of modern open source software (OSS) development principles for generating reproducible scientific publications. Concretely, we make the case for treating an article as an OSS project following a DevOps approach and applying software engineering best-practices to manage its associated artifacts and maintain the reproducibility of its findings. Popper leverages existing cloud-computing infrastructure and DevOps tools to produce academic articles that are easy to validate and extend. We present a use case that illustrates the usefulness of this approach. We show how, by following the *Popper* convention, reviewers and researchers can quickly get to the point of getting results without relying on the original author's intervention.**

## I. Introduction

A key component of the scientific method is the ability to revisit and replicate previous experiments. Managing information about an experiment allows scientists to interpret and understand results, as well as verify that the experiment was performed according to acceptable procedures. Additionally, reproducibility plays a major role in education since the amount of information that a student has to digest increases as the pace of scientific discovery accelerates. By having the ability to repeat experiments, a student learns by looking at provenance information about the experiment, re-evaluates the questions that the original experiment addressed and builds upon the results of the original study.

Independently validating experimental results in the field of computer systems research is a challenging task [1,2]. Recreating an environment that resembles the one where an experiment was originally executed is a time-consuming endeavour [3,4]. In this work, we revisit the idea of an executable paper [5,6], which proposes the integration of executables and data with scholarly articles to help facilitate its reproducibility. Our approach is to implement it in today's cloud-computing world by treating an article as an open source software (OSS) project.

We introduce *Popper*, a convention for systematically implementing the different stages of the experimentation workflow following a DevOps [7] approach. The convention can be summarized in three high-level guidelines:

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

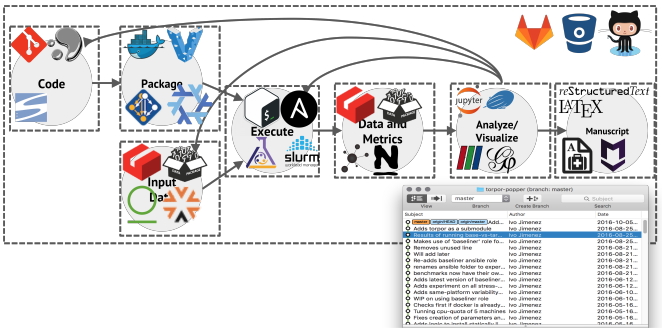


Figure 1: A generic experimentation workflow viewed through a DevOps looking glass.

By following these guidelines, researchers can make all artifacts associated to an article publicly available with the goal of maximizing automation when an experiment is re-executed and results are validated. This paper describes our experiences with the Popper Convention which we have successfully followed to aid in producing papers and classroom lessons that are easy to reproduce. This paper makes the following contributions:

- An analysis of how the DevOps practice can be repurposed to an academic article (Section II and Section III);
- Popper: A methodology for writing academic articles and associated experiments following the DevOps practice (Section IV);
- Popper-CLI: an experiment bootstrapping tool that makes Popper-compliant experiments readily available to researchers; and

- A use case that details how to follow the convention and illustrates its benefits (Section V).

In this work, we demonstrate the benefits of following the Popper convention: it brings order to personal research workflows; lowers the barrier for others to re-execute published experiments on multiple platforms with minimal effort and without having to speculate on what the original authors went through; and leverages automated regression testing to maintain the reproducibility integrity of experiments.

## II. Experimental Practices

In this section we examine common practices and identify desired features for a new experimental methodology.

### A. Common Practice

*1) Ad-hoc Personal Workflows:* A typical practice is the use of custom bash scripts to automate some of the tasks of executing experiments and analyzing results. From the point of view of researchers, having an ad-hoc framework results in more efficient use of their time, or at least that's the belief. Since these are personalized scripts, they usually hard-code many of the parameters or paths to files in the local machine. Worst of all, a lot of the contextual information is in the mind of researchers. Without a list of guiding principles, going back to an experiment, *even for the original author on the same machine*, represents a time-consuming task.

*2) Sharing Source Code:* Version-control systems give authors, reviewers and readers access to the same code base but the availability of source code does not guarantee reproducibility [3]; code may not compile, and even it does, results may differ due to differences from other components in the software stack. While sharing source code is beneficial, it leaves readers with the daunting task of recompiling, reconfiguring, deploying and re-executing an experiment. Things like compilation flags, experiment parameters and results are fundamental contextual information for re-executing an experiment.

*3) Experiment Repositories:* An alternative to sharing source code is experiment repositories [8,9]. These allow researchers to upload artifacts associated with a paper, such as input data sets. Similar to code repositories, one of the main problems is the lack of automation and structure for the artifacts. The availability of the artifacts does not guarantee the reproduction of results since a significant amount of manual work needs to be done after these have been downloaded. Additionally, large data dependencies cannot be uploaded since there is usually a limit on the artifact file size.

*4) Virtual Machines:* A Virtual Machine (VM) can be used to partially address the limitations of only sharing source code. However, in the case of systems research where the performance is the subject of study, the overheads in terms of performance (the hypervisor "tax") and management (creating, storing and transferring) can be high and, in some cases, they cannot be accounted for easily [10]. In scenarios where OS-level virtualization is a viable alternative, it can be used instead of hardware-level virtualization [11].

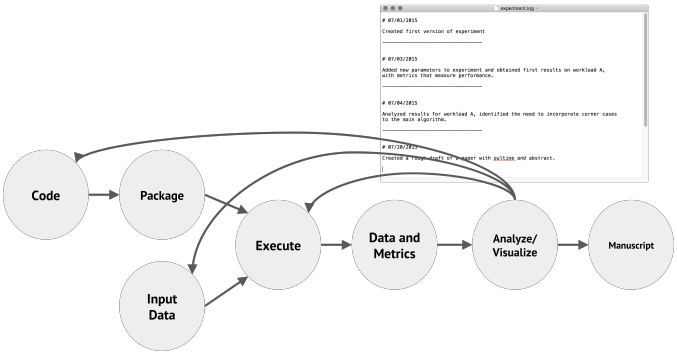


Figure 2: A generic experimentation workflow typically followed by researchers in projects with a computational component. Some of the reasons to iterate (backwards-going arrows) are: fixing a bug in the code of a system, changing a parameter of an experiment or running the same experiment on a new workload or compute platform. Although not usually done, in some cases researchers keep a chronological record on how experiments evolve over time (the analogy of the lab notebook in experimental sciences).

*5) Experiment Packing:* Experiment packing entails tracing an experiment at runtime to capture all its dependencies and generating a package that can be shared with others [12,13]. Experiment packing is an automated way of creating a virtual machine or environment and thus suffers from the same limitations: external dependencies such as large datasets cannot be packaged; the experiment is a black-box without contextual information (e.g. history of modifications) that is hard to introspect, making difficult to build upon existing work; and packaging does not explicitly capture validation criteria.

*6) Data Analysis Ad-hoc Approaches:* A common approach to analyze data is to capture CSV files and manually paste their contents into Excel or Google Spreadsheets. This manual manipulation and plotting lacks the ability to record important steps in the process of analyzing results, such as the series of steps needed to go from a CSV to a figure. Even if the associated spreadsheet is made available, it is not immediately clear what a researcher did.

*7) Eyeball Validation:* Assuming the reader is able to recreate the environment of an experiment, validating the outcome requires domain-specific expertise in order to determine the differences between original and recreated environments that might be the root cause of any discrepancies in the results [1,14]. Additionally, reproducing experimental results when the underlying hardware environment changes is challenging mainly due to the inability to predict the effects of such changes in the outcome of an experiment [15]. In this case validation is typically done by "eyeballing" figures and the description of experiments in a paper, a subjective task, based entirely on the intuition and expertise of domain-scientists.

### B. Goals for a New Methodology

A diagram of a generic experimentation workflow is shown in Fig. 2. The problem with current practices is that each

of them partially cover the workflow. For example, sharing source code only covers the first task (source code); experiment packing only covers the second one (packaging); and so on. Based on this, we see the need for a new methodology that:

- Is reproducible without incurring any extra work for the researcher. It should require the same or less effort than current practices with the difference of doing things in a systematic way.
- Improves the personal workflows of scientists by having a common methodology that works for as many projects as possible and that can be used as the basis of collaboration.
- Captures the end-to-end workflow in a modern way, including the history of changes that are made to an article throughout its lifecycle.
- Makes use of existing tools (do not reinvent the wheel!).
- Has the ability to handle large datasets.
- Captures validation criteria in an explicit manner so that subjective evaluation of results of a re-execution is minimized.
- Results in experiments that are amenable to improvement and allows easy collaboration, as well as making it easier to build upon existing work.

## III. The DevOps Toolkit

When we compare the generic experimentation workflow shown in Fig. 2 against best-practices in the software engineering world, and in open source communities in particular, we observe a very close correspondence (Tbl. I).

Table I: Comparison of practices in a scientific exploration against those in software projects.

| Scientific Exploration | Software Project |
|---|---|
| Experiment code | Source code |
| Input data | Test examples |
| Analysis and Visualization | Test analysis |
| Validation | Regression testing |
| Manuscript / notebook | Documentation / reports |

In recent years, the term DevOps [7] has been used to refer to these set of common practices, which have the goal of expediting the delivery of a software project, allowing to iterate as fast as possible on improvements and new features, without undermining the quality of the product. In our work, we make the case for achieving the same outcome by treating an academic article as the "software product" being delivered.

DevOps puts an emphasis on versioning every dependency of a software project, from infrastructure to any runtime requirement, with the goal of executing pipelines by providing a list of these versioned artifacts to DevOps tools. In the DevOps world, this is referred to as having "infrastructure-as-code" and basically "anything-as-code" [16]. In practice, this means typing commands in a script file, instead of directly on the CLI, and letting automation tools execute them. These scripts are then version-controlled in order to keep track of changes in a systematic way.

In this section we review and highlight salient features of the DevOps toolkit that makes it amenable to organize all artifacts associated with an academic article. To guide our discussion, we refer to the generic experimentation workflow viewed through a DevOps looking glass shown in Fig. 1 (the same workflow as in Fig. 2 with DevOps tools overlayed on top[1]). In Section IV we analyze more closely the composability of these tools and describe general guidelines (the convention).

### A. Version Control

Traditionally the content managed in a version-control system (VCS) is the project's source code; for an academic article the equivalent is the article's content: article text, experiments (code and data) and figures. The idea of keeping an article's source in a VCS is not new and in fact many people follow this practice [6]. However, this only considers automating the generation of the article in its final format (usually PDF). While this is useful, here we make the distinction between changing the prose of the paper, changing the parameters of the experiment (both its components and its configuration), as well as storing the experiment results.

Ideally, one would like the entire end-to-end pipeline for all the experiments contained in an article to be managed by a VCS. With the advent of cloud-computing, this is possible for most research articles[2]. In a sense, having all the article's dependencies in the same repository is analogous to how large cloud companies maintain monolithic repositories to manage their internal infrastructure [17,18] but at a lower scale.

**Tools and services**: Git, Svn and Mercurial are popular VCS tools. GitHub, GitLab and BitBucket are web-based Git repository hosting services. They offer all of the distributed revision control and VCS functionality of Git as well as adding their own features. The entire history of the project and its artifacts of public repositories can be browsed online.

### B. Package Management

Availability of code does not guarantee reproducibility of results [3]. The second main component in the experimentation pipeline is the packaging of applications so that users do not have to do it themselves. Software containers (e.g. Docker, OpenVZ or FreeBSD's jails) complement package managers by packaging all the dependencies of an application in an entire file system snapshot that can be deployed in systems "as is" without having to worry about problems such as package dependencies or specific OS versions. From the point of view of an academic article, these tools can be leveraged to package the dependencies of an experiment. Software containers like

[1]Logos in Fig. 1 correspond to commonly used tools from the DevOps toolkit. From left-to-right, top-to-bottom: Git, Mercurial, Subversion (code); Docker, Vagrant, Spack, Nix (packaging); Git-LFS, Datapackages, Artifactory, Archiva (input data); Bash, Ansible, Puppet, Slurm (execution); Git-LFS, Datapackages, Icinga, Nagios (output data and runtime metrics); Jupyter, Zeppelin, Paraview, Gephi (analysis and visualization); RestructuredText, LATeX, Asciidoc and Markdown (manuscript); GitLab, Bitbucket and GitHub (experiment changes and labnotebook functionality).

[2]For large-scale experiments or those that run on specialized platforms, re-executing an experiment might be difficult. However, this does not exclude such research projects from being able to keep the article's associated assets under version control.

Docker have the great potential for being of great use in computational sciences [19].

**Tools and services**: Docker automates the deployment of applications inside software containers by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux. Alternatives to Docker are modern package managers such as Nix or Spack, or, in the case of virtual machines, Vagrant.

### C. Experiment Orchestration And Environment Capture

Experiments that require a set of machines to be orchestrated can make use of a tool that automatically drives the end-to-end execution of the experiment. This category of tools can also serve for capturing the details about the runtime environment (e.g. hardware and OS configuration, versions of system libraries, etc.). Associating this information with experiment results allows to make environmental comparisons across multiple executions [20].

**Tools and services**: Ansible is a configuration management utility for configuring and managing computers, as well as deploying and orchestrating multi-node applications. Similar tools include Puppet, Chef, Salt, among others.

### D. Infrastructure Automation

When on-premises infrastructure is not available (e.g. when executing on cloud platforms), a researcher has to deal with requesting, configuring and provisioning infrastructure. In these scenarios, tooling exists to automate these steps on remote computational resources.

**Tools and services**: Cloudlab, Chameleon, PRObE and XSEDE are NSF-sponsored infrastructures that allows users to request computing resources to execute multi-node experiments. Additionally, public cloud service providers such as Amazon, Google, Packet and Rackspace allow users to deploy applications on virtual and bare-metal instances. Terraform is a tool that allows to automate the configuration and provisioning of infrastructure in a platform-agnostic way. When a Terraform provider for a particular infrastructure is not available, one can resort to using platform-specific tools directly. For example CloudLab [21], Grid500K [22], AWS CloudFormation and OpenStack Heat provide CLI tools for this purpose.

### E. Dataset Management

Some experiments involve the processing of large input, intermediary or output datasets. While possible, traditional VCS tools such as Git were not designed to store large binary files. A proper artifact repository client or dataset management tool can take care of handling data dependencies.

**Tools and services**: Examples are Apache Archiva, Git-LFS, Datapackages or Artifactory.

### F. Data Analysis and Visualization

Once an experiment runs, the next task is to analyze and visualize results. Many tools can support this task, as long as a user can script the analysis and automate its execution, any tool can fit in this category.

**Tools and services**: Jupyter notebooks run as a web-based application. It facilitates the sharing of documents containing live code (in Julia, Python or R). Binder is service that allows one to turn a GitHub repository into a collection of interactive Jupyter notebooks so that readers do not need to deploy web servers themselves. Alternatives to Jupyter are Gnuplot, Zeppelin and Beaker. Other scientific visualization such as Paraview and Gephi tools can also fit in this category.

### G. Performance Monitoring

Prior to and during the execution of an experiment, capturing performance metrics can be beneficial. In the case of systems research articles, where performance is the main subject of study, capturing performance metrics is fundamental. Instead of creating ad-hoc tools to achieve this one can adopt and extend existing tools. At the end of the execution, the captured data can be analyzed and many of the graphs included in the article can come directly from running analysis scripts on top of this data.

**Tools and services**: Many mature monitoring tools exist such as Nagios, Ganglia, StatD, CollectD, among many others. For measuring single-machine baseline performance, tools like Conceptual (network), stress-ng (CPU, memory), fio and many others exist.

### H. Continuous Integration

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository frequently with the purpose of catching errors as early as possible. The experiments associated with an article can also benefit from CI. If an experiment's findings can be codified in the form of a unit test, this can be verified on every change to the article's repository.

**Tools and services**: Travis CI is an open-source, hosted, distributed continuous integration service used to build and test software projects hosted at GitHub. Alternatives to Travis CI are CircleCI and CodeShip. Other self-hosted solutions exist such as Jenkins.

### I. Automated Performance Regression Testing

Open source projects such as the Linux kernel go through rigorous performance testing to ensure that newer versions do not introduce any problems. Performance regression testing is usually an ad-hoc activity but can be automated using high-level languages [23] or statistical techniques [24].

**Tools and services**: Language-specific frameworks for validating performance such as ScalaMeter exist. Aver is a generic language and validation tool that allows users to express and corroborate statements about runtime metrics gathered.

## IV. The Popper Convention: A DevOps Approach to Producing Academic Papers

The goal for *Popper* is to give researchers a common framework to reason, in a systematic way, about how to structure all the dependencies and generated artifacts associated with an experiment. The convention can be summarized in the

**Listing 1** Sample contents of a Popper repository.

```
paper-repo
| README.md
| .popper.yml
| experiments
|   |-- myexp
|   |   |-- datasets/
|   |   |     |-- input-data.csv
|   |   |-- figure.png
|   |   |-- process-result.py
|   |   |-- setup.yml
|   |   |-- results.csv
|   |   |-- run.sh
|   |   |-- validations.aver
|   |    -- vars.yml
| paper
|   |-- build.sh
|   |-- figures/
|   |-- paper.tex
|    -- references.bib
```

three high-level steps outlined in Section I. These guidelines provide the following unique features:

1. Provides an experimentation protocol for generating self-contained experiments.
2. Makes it easier for researchers to explicitly specify validation criteria.
3. Abstracts domain-specific experimentation workflows and toolchains.
4. Provides reusable experiment templates that provide curated experiments commonly used by a research community.

### A. Self-containment

We say that an experiment is Popper-compliant (or that it has been "Popperized") if all of the following is available, either directly or by reference, in one single source-code repository: experiment code, experiment orchestration code, reference to data dependencies, parametrization of experiment, validation criteria and results. In other words, a Popper repository contains all the dependencies for an article, including its manuscript.

An example paper project is shown in Lst. 1. A paper repository is composed primarily of the article text and experiment orchestration logic. The actual code that gets executed by an experiment is not part of the repository. This, as well as any large datasets that are used as input to an experiment, resides in their own repositories and are stored in the experiment folder of paper repository as references.

With all these artifacts available, the reader can easily deploy an experiment or rebuild the article's PDF that might include new results. Fig. 3 shows our vision for the reader/reviewer workflow when reading a Popper for a Popperized article. The diagram uses tools we use in the use-case in Section 5.2, like Ansible and Docker, but as mentioned earlier, these can be swapped by equivalent tools. Using this workflow, the writer is completely transparent and the article consumer is free to explore results, re-run experiments, and contradict assetions in the paper.

A paper is written in any desired markup language. In the above listing we use LATeX as an example (`paper.tex` file).

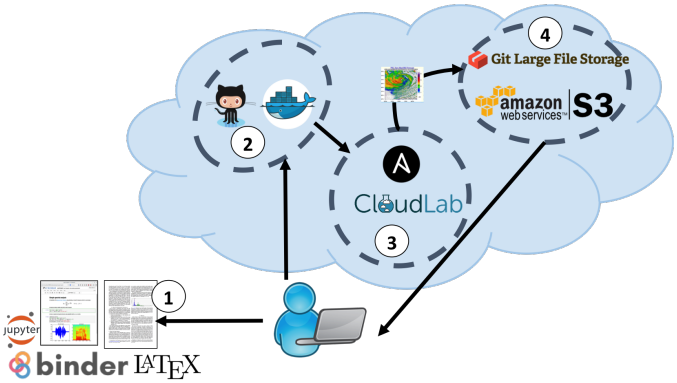


Figure 3: A sample workflow a paper reviewer or reader would use to read a Popperized article. (1) The PDF, Jupyter or Binder are used to visualize and interact with the results post-mortem on the reader's local machine (2) If needed the reader has the option of looking at the code and clone it locally (GitHub); for single-node experiments, they can be deployed locally too (Docker) (3) For multi-node experiments, Ansible can then be used to deploy the experiment on a public or private cloud (NSF's CloudLab in this case) (4) Lastly, experiments producing large data sets can make use of cloud storage.

There is a `build.sh` command that generates the output format (e.g. `PDF`). For the experiment execution logic, each experiment folder contains the necessary information such as setup, output post-processing (data analysis) and scripts for generating an image from the results. The execution of the experiment will produce output that is either consumed by a post-processing script, or directly by the scripts that generate an image.

The output can be in any format (CSVs, HDF, NetCDF, etc.), as long as it is versioned and referenced. An important component of the experiment logic is that it should assert the original assumptions made about the environment (a `setup.yml` file in the example), for example, the operating system version (if the experiment assumes one). Also, it is important to parametrize the experiment explicitly (e.g. `vars.yml`), so that readers can quickly get an idea of what is the parameter space of the experiment and what they can modify in order to obtain different results. One common practice we follow is to place in the caption of every figure a `[source]` link that points to the URL of the corresponding post-processing script in the version control web interface (e.g. GitHub).

### B. Automated Validation

Validation of experiments can be classified in two categories. In the first one, the integrity of the experimentation logic is checked using existing continuous-integration (CI) services such as TravisCI, which expects a `.travis.yml` file in the root folder. This file contains a specification that consists of a list of tests that get executed every time a new commit is added to the repository. These types of checks can verify that the paper is always in a state that can be built (generate the PDF correctly); that the syntax of orchestration files is correct

so that if changes occur, e.g., addition of a new variable, it can be executed without any issues; or that the post-processing routines can be executed without problems.

The second category of validations is related to the integrity of the experimental results. These domain-specific tests ensure that the claims made in the paper are valid for every re-execution of the experiment, analogous to performance regression tests done in software projects that can be implemented using the same class of tools. Alternatively, claims can also be corroborated as part of the analysis code. When experiments are not sensitive to the effects of virtualized platforms, these assertions can be executed on public/free CI platforms (e.g. TravisCI runs tests in VMs). However, when results are sensitive to the underlying hardware, it is preferable to leave this out of the CI pipeline and make them part of the post-processing routines of the experiment. In the example above, an `assertions.aver` file contains validations in the Aver [23] language that check the integrity of runtime performance metrics that claims make reference to. Examples of these type of assertions are: "the runtime of our algorithm is 10x better than the baseline when the level of parallelism exceeds 4 concurrent threads"; or "for dataset A, our model predicts the outcome with an error of 95%".

When validating assertions that depend on the underlying hardware, i.e. that come from capturing runtime performance metrics, an important step is to corroborate that the baseline performance of the experiment for a new environment can be reproduced. While this is a similar test that can be codified using performance regression testing as mentioned in the above paragraph, we make the distinction since this step can be executed before any experiment runs. If the baseline performance cannot be reproduced, there is no point in executing the experiment. Many of the commonly used orchestration tools incorporate functionality for obtaining "facts" about the environment, information that is useful to have when corroborating assumptions; other monitoring tools such as Nagios can capture raw system-level performance; and existing frameworks such as baseliner are designed to obtain baseline profiles that are associated to experimental results.

### C. Toolchain Agnosticism

We conceived Popper as a general convention, applicable to a wide variety of environments, from cloud to high-performance computing (HPC). In general, Popper can be applied in any scenario where a component (data, code, infrastructure, hardware, etc.) can be referenced by an identifier, and where there is an underlying tool that consumes these identifiers so that they can be acted upon (install, run, store, visualize, etc.). The core idea behind Popper is to borrow from the DevOps movement [16] the idea of treating every component as an immutable piece of information and provide references to scripts and components for the creation, execution and validation of experiments (in a systematic way) rather than leaving to the reader the daunting task of inferring how binaries and experiments were generated or configured.

**Listing 2** Initialization of a Popper repo.

```
$ cd mypaper-repo
$ popper init
-- Initialized Popper repo

$ popper experiment list
-- available templates ---------------
ceph-rados         proteustm  mpi-comm-variability
cloverleaf         gassyfs    zlog
spark-standalone   torpor     malacology

$ popper add torpor myexp
```

We say that a tool is Popper-compliant if it has the following two basic properties:

1. Assets can be be associated to unique identifiers. Code, packages, configurations, data, results, etc. can all be referenced and uniquely identified.
2. The tool is scriptable (e.g. can be invoked from the command line) and can act upon given asset IDs.

In Section III we provided a list of tools for every category of the generic experimentation workflow (Fig. 2) that comply with the two properties given above. In general, tools that are hard to script e.g. because they do not provide a command-line interface (can only interact via GUI) or they only have a programmatic API for a non-interpreted language, are beyond the scope of Popper.

### D. Experiment Templates

Researchers that decide to follow Popper are faced with a steep learning curve, especially if they have only used a couple of tools from the DevOps toolkit. To lower the entry barrier, we have developed a command line interface (CLI) tool to help bootstrap a paper repository that follows the Popper convention. As part of our efforts, we maintain a list of experiment templates that have been "Popperized". These are end-to-end experiments that use a particular toolchain and for which execution, production of results and generation of figures has been implemented (see Section V for examples; each use case is available as an experiment in the templates repository). The CLI tool can list and show information about available experiments. Assuming a git repository has been initialized, the tool allows to add experiments to the repository (Lst. 2). Templates and CLI can be found at http://falsifiable.us.

## V. Use Case: Quantifying the Scalability of an In-memory File System

We now show a use case that illustrates the usefulness of the Popper convention. We refer the reader to the Popper website to find other use cases and examples of articles following the convention. Additionally, the Popper repository of this article (https://github.com/systemslab/popper-paper) contains more detailed information about the experimental setup.

GassyFS [25] is a new prototype file system that stores files in distributed remote memory and provides support for checkpointing file system state. The core of the file system is a user-space library that implements a POSIX file interface. File

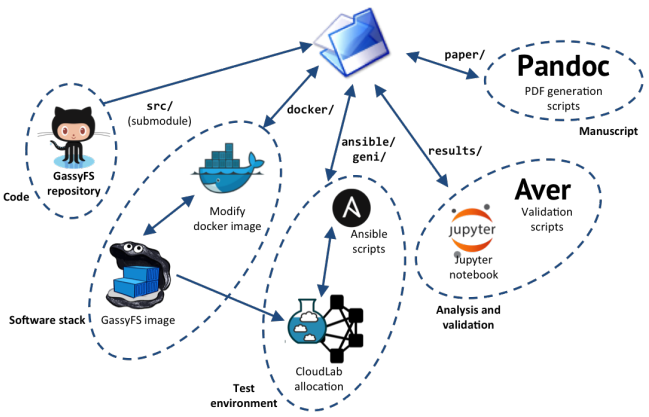


Figure 4: Popper workflow for the GassyFS experiment.

system metadata is managed locally in memory, and file data is distributed across a pool of network-attached RAM managed by worker nodes and accessible over RDMA or Ethernet. Applications access GassyFS through a standard FUSE mount, or may link directly to the library to avoid any overhead that FUSE may introduce. By default all data in GassyFS is non-persistent. That is, all metadata and file data is kept in memory, and any node failure will result in data loss. In this mode GassyFS can be thought of as a high-volume `tmpfs` that can be instantiated and destroyed as needed, or kept mounted and used by applications with multiple stages of execution. The differences between GassyFS and `tmpfs` become apparent when we consider how users deal with durability concerns.

### A. Popperizing The Experiment

In this experiment we aim to evaluate the scalability of GassyFS, i.e. how it performs when we increase the number of nodes in the underlying GASNet-backed FUSE mount. For this experiment, we select Docker for packaging the software stack; Ansible for orchestrating the experiment; Geni-lib for requesting infrastructure on CloudLab; Jupyter for visualizing results; and Aver to validate results. The instantiation of the generic experimentation workflow is shown in Fig. 4. The contents of the folder are shown in Lst. 3. We have a subfolder for each DevOps tool we use, and corresponding scripts in them. The `docker` folder contains the files needed to recreate the image for GassyFS. The repository that hosts the GassyFS source code is a referenced via git submodule (not shown). Other tools used in the experiment (Ansible, geni-lib, Aver, Jupyter and Pandoc) are also containerized (not shown).

### B. Codifying Expectations

In Fig. 5 we show results for the experiment. We note that while the actual numbers obtained are relevant, they are not our main focus. Instead, we put more emphasis on the goals of the experiments, how we can reproduce results on multiple platforms with minimal effort, and how we can ensure the validity of the results. Fig. 5 shows the results of compiling Git on GassyFS. We observe that once the cluster gets to 2 nodes, performance degrades sublinearly with the number

**Listing 3** Contents of the GassyFS experiment.

```
paper-repo/experiments/gassyfs
| README.md
| ansible
|   |-- ansible.cfg
|   |-- machines
|   |-- playbook.yml
|   |-- workloads/
|   |   |-- compile.yml
|   |   |-- git.yml
|   |   |-- kernel.yml
| docker
|   |-- gassyfs/
|   |   |-- Dockerfile
|   |   |-- entrypoint.sh
| geni
|   |-- cloudlab_request.py
| results
|   |-- output.csv
|   |-- visualize.ipynb
| run.sh
| setup.sh
| validate.sh
| vars.yml
```

of nodes. This is expected for workloads such as the one in question. The Aver assertion in Lst. 4 is used to check the integrity of this result. This expresses our expectation of GassyFS performing sublinearly with respect to the number of nodes. After the experiment runs, Aver is invoked to test this statement against the experiment results obtained.

**Listing 4** Assertion to check the scalability of GassyFS.

```
when
  workload=* and machine=*
expect
  sublinear(nodes,time)
```

### C. Re-executing The Experiment (The Value of Popper)

This experiment runs on remote infrastructure and is driven from wherever the repository is checked out as shown in Fig. 4 (e.g. a user's laptop). The only dependency on local and remote nodes is Bash and Docker. Re-executing this experiment on any CloudLab site is trivial. The `vars.yml` file contains parameters for specifying the site where the experiment is deployed, as well as credentials needed by the `geni-lib` CLI to authenticate with CloudLab. Supporting other platforms can be achieved by using one of the tools mentioned in Section III-D. For example, Terraform can be used to request Docker/Linux nodes on public clouds. Lst. 5 shows a snippet for requesting a node on DigitalOcean. This file would then reside in a `terraform` folder and would be passed to the Terraform CLI tool. Besides including options related to infrastructure, the `vars.yml` file also makes other experiment variables available to the re-executioner. Parameters such as the workloads executed, number of nodes in the GassyFS cluster and GASNet/FUSE configuration can be modified in order to observe the effect these have on the performance of the system.

Although GassyFS is simple in design, it is relatively complex to setup. The combinatorial space of possible ways in

**Listing 5** Terraform configuartion for requesting a Droplet.

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

which the system can be compiled, packaged and configured is large. For example, the version of GCC we use (4.9) has approximately $10^8$ possible ways of compiling a binary. In GASNet 2.6, there are 64 flags for additional packages and 138 flags for additional features. To mount GassyFS, we use FUSE, which can be given more than 30 different options, many of them taking multiple values.

As shown in this use case, execution and validation of results is fully automated. By contrasting this with the common practice of providing source code and a `README` that a reader has to go through, we can perceive the benefits of Popperizing experiments. Additionally, having *all* dependencies and parametrization explicitly available eases the "reverse walking" of the experimentation workflow, starting with a PDF, going back to a Jupyter notebook to see the analysis of results, and possibly drilling down all the way to the source code.

## VI. Discussion

### A. Limitations

Traditional experimentation practices are deeply rooted in the muscle memory of researchers, typing commands in "live" systems and getting results as they go. The use case presented in the previous section illustrates how Popper allows a researcher to have a systematic approach to automating experiments. While it might seem like a burden at the beginning of an experimental exploration, following Popper quickly pays-off. Consider the common situation of going back to an experiment after a short amount of time and how we struggle when we try to remember what was done, or why things were done in a particular way.

However, Popper is not perfect. Obvious issues such as the lack of resources, either due to the use of special hardware or the unavailability of extreme-scale allocations, have to be resolved before a Popperized experiment is re-executed. Additionally, in some cases, the choice of a tool might affect the validation of results, for example such as in cases where VMs introduce ineligible overheads.

### B. DevOps Skills as Professional Development

While the learning curve for the DevOps toolkit is steep, having these as part of the skillset of students or researchers-in-training can only improve their curriculum. Since industry as well as many industrial and national laboratories have embraced the DevOps practice (or are in the process of embracing it), making use of these tools improves their prospects of future employment. In our view, the DevOps toolkit is analogous to those employed by scientists in other domains (e.g. a bioscience toolkit) and, as such, in order to be successful as a researcher, one has to master their use.

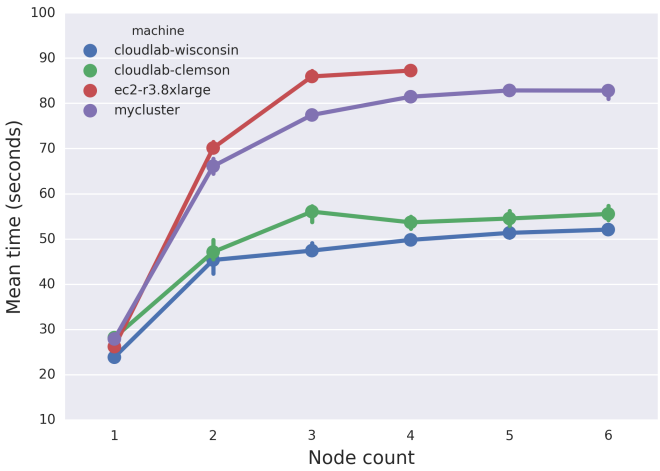


Figure 5: Scalability of GassyFS as the number of nodes in the GASNet cluster increases. The workload in question compiles Git. (source: goo.gl/gd15XN).

### C. Automated Reproducibility Validation

As mentioned earlier, continuous integration (CI) services and tools can be used to ensure that the artifacts associated to an article are in "good shape", i.e. that the software can be built successfully, experiments re-executed without errors and results obtained corroborate original results [26]. Adapting existing CI solutions to check the integrity of Popperized repositories can be time-consuming. To avoid this, one can organize the contents of the repository following the common structure introduced in [27], which allows to automate the end-to-end execution of Popperized experiments.

### D. Popperized Experiments as Experiment Packages

Our vision is that, over time, as more experiments become Popperized and aggregated in the form of Popper template repositories, these can become analogous to software packages that are currently used in the open source software community. With such a list of experiments for a particular community, these experiments then can be indexed so that when a student or researcher looks for preliminary work, they can get to existing, reproducible experiments that they can use as the basis of their work.

### E. Popper Complements Existing Efforts

There have been efforts to address reproducibility issues in subdomains of the systems research community. We believe Popper complements many of these since it encourages a practice (i.e. to follow a protocol) that applies on top of tools that researchers already know rather than requiring scientists to learn a whole new suite of tools. The following are some examples of community efforts and projects where Popper can complement:

- Ctuning Foundation's Extended Artifact Description Guide[3] is a set of high-level guidelines for authors on how to prepare an "Artifact Evaluation" appendix for academic articles. Conferences such as Supercomputing, TRUST@PLDI, CGO/PPoP and others are currently making use of it for their reproducibility initiatives. Popper implements a similar pipeline as the one described in the Artifact Description Guide. A Popper repository could even be used instead of an "Artifact Evaluation" appendix.
- Elsevier's 2011 Executable Paper Challenge[4] gave the first prize to the Collage Authoring Environment [28]. Popper is an alternative that makes use of the DevOps toolkit, allowing researchers to keep using their tools but to structure their explorations in a systematic way.
- Proxy applications (Mini-apps) in HPC [29] can be accompanied with a Popper repository to make it easier to validate performance results and facilitate the execution of these on different platforms.
- The Open Encyclopedia of Parallel Algorithmic Features [30] could have a set of Popper repositories, (one per each article), making it easier for readers to reuse the algorithms and their insights.
- The Journal of Information Systems has recently adopted a new publication model that incentivizes reproducibility by inviting original authors to collaborate with independent reviewers and publish a subsequent paper on whether they could reproduce the original work [31]. By following Popper authors can reduce the amount of work that these subsequent publications entail.
- ReScience[5] is a peer-reviewed journal that targets computational research and encourages the explicit replication of already published research. Popperizing these replications can have the benefit of allowing others to easily re-execute experiments.

## VII. Conclusion and Future Work

By following the Popper convention, researchers can reduce significantly the time that others spend re-executing their experiments, making it easier to share and collaborate with others. We are currently working with researchers from other domains such as numeric weather prediction [32] and mathematical sciences [33] to automate experiments that follow the Popper convention. While Popper facilitates the re-execution of experiments, it cannot serve for identifying root causes of irreproducibility. An open problem is to automatically identify sources of irreproducibility, either from changes made to an experiment or from changes in the environment

**Acknowledgments**: This work was partially funded by the Center for Research in Open Source Software[6], Sandia National Laboratories and NSF Award #1450488. A short version of this article appears in [34].

[3] http://ctuning.org/ae/submission_extra.html
[4] http://www.executablepapers.com
[5] https://rescience.github.io
[6] http://cross.ucsc.edu

## VIII. Bibliography

[1] J. Freire, P. Bonnet, and D. Shasha, "Computational Reproducibility: State-of-the-art, Challenges, and Database Research Opportunities," *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012.

[2] G. Fursin, "Collective Mind: Cleaning up the research and experimentation mess in computer engineering using crowdsourcing, big data and machine learning," *arXiv:1308.2410 [cs, stat]*, 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, 2016, pp. 62–69.

[4] T. Hoefler and R. Belli, "Scientific Benchmarking of Parallel Computing Systems: Twelve Ways to Tell the Masses when Reporting Performance Results," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015.

[5] R. Strijkers, R. Cushing, D. Vasyunin, C. de Laat, A.S.Z. Belloum, and R. Meijer, "Toward Executable Scientific Publications," *Procedia Computer Science*, vol. 4, 2011.

[6] M. Dolfi, J. Gukelberger, A. Hehn, J. Imriska, K. Pakrouski, T.F. Rønnow, M. Troyer, I. Zintchenko, F.S. Chirigati, J. Freire, and D. Shasha, "A Model Project for Reproducible Papers: Critical Temperature for the Ising Model on a Square Lattice," *arXiv:1401.2000 [cond-mat, physics:physics]*, vol. abs/1401.2000, Jan. 2014. Available at: http://arxiv.org/abs/1401.2000.

[7] G. Kim, J. Humble, P. Debois, and J. Willis, *The DevOps Handbook*, O'Reilly Media, 2016.

[8] V. Stodden, S. Miguez, and J. Seiler, "ResearchCompendia.org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science," *Computing in Science & Engineering*, vol. 17, Jan. 2015.

[9] D.D. Roure, C. Goble, and R. Stevens, "Designing the myExperiment Virtual Research Environment for the Social Sharing of Workflows," *IEEE International Conference on e-Science and Grid Computing*, 2007.

[10] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J.N. Matthews, "Xen and the Art of Repeated Research," *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, 2004.

[11] I. Jimenez, C. Maltzahn, J. Lofstead, A. Moody, K. Mohror, R.H. Arpaci-Dusseau, and A. Arpaci-Dusseau, "The Role of Container Technology in Reproducible Computer Systems Research," *2015 IEEE International Conference on Cloud*

*Engineering (IC2E)*, 2015.

[12] F. Chirigati, R. Rampin, D. Shasha, and J. Freire, "ReproZip: Computational Reproducibility with Ease," *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*, 2016.

[13] A.P. Davison, M. Mattioni, D. Samarkanov, and B. Telenczuk, "Sumatra: A Toolkit for Reproducible Research," *Implementing Reproducible Research*, 2014.

[14] 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.

[15] R.H. Saavedra-Barrera, "CPU Performance Evaluation and Execution Time Prediction Using Narrow Spectrum Benchmarking," PhD thesis, EECS Department, University of California, Berkeley, 1992.

[16] A. Wiggins, "The Twelve-Factor App," *The Twelve-Factor App*, 2011.

[17] C. Tang, T. Kooburat, P. Venkatachalam, A. Chander, Z. Wen, A. Narayanan, P. Dowell, and R. Karl, "Holistic Configuration Management at Facebook," *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015.

[18] C. Metz, "Google Is 2 Billion Lines of Code's All in One Place," *WIRED*, Sep. 2015.

[19] C. Boettiger, "An introduction to Docker for reproducible research, with examples from the R environment," *arXiv:1410.0846 [cs]*, Oct. 2014. Available at: http://arxiv.org/abs/1410.0846.

[20] I. Jimenez, C. Maltzahn, J. Lofstead, A. Moody, K. Mohror, R. Arpaci-Dusseau, and A. Arpaci-Dusseau, "Tackling the Reproducibility Problem in Storage Systems Research with Declarative Experiment Specifications," *Proceedings of the 10th Parallel Data Storage Workshop*, 2015.

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

[22] 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.

[23] I. Jimenez, C. Maltzahn, 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.

[24] T.H. Nguyen, B. Adams, Z.M. Jiang, A.E. Hassan, M. Nasser, and P. Flora, "Automated Detection of Performance Regressions Using Statistical Process Control Techniques," *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, 2012.

[25] N. Watkins, M. Sevilla, and C. Maltzahn, *GassyFS: An In-Memory File System That Embraces Volatility, UCSC-SOE-16-08*, UC Santa Cruz, 2016.

[26] J. Howison, "Retract bit-rotten publications: Aligning incentives for sustaining scientific software," *Second Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE2)*, Jul. 2014.

[27] I. Jimenez, A. Arpaci-Dusseau, R. Arpaci-Dusseau, J. Lofstead, C. Maltzahn, K. Mohror, and R.P. Ricci, "PopperCI: automated reproducibility validation," *2017 IEEE INFOCOM International Workshop on Computer and Networking Experimental Research Using Testbeds*, Atlanta, USA: 2017.

[28] P. Nowakowski, E. Ciepiela, D. Harężlak, J. Kocot, M. Kasztelnik, T. Bartyński, J. Meizner, G. Dyk, and M. Malawski, "The collage authoring environment," *Procedia Computer Science*, vol. 4, 2011.

[29] S. Dosanjh, R. Barrett, M. Heroux, and A. Rodrigues, "Achieving Exascale Computing through Hardware/Software Co-design," *Recent Advances in the Message Passing Interface*, 2011.

[30] V. Voevodin, A. Antonov, and J. Dongarra, "AlgoWiki: An open encyclopedia of parallel algorithmic features," *Supercomputing frontiers and innovations*, vol. 2, 2015.

[31] F. Chirigati, R. Capone, R. Rampin, J. Freire, and D. Shasha, "A collaborative approach to computational reproducibility," *Information Systems*, Mar. 2016.

[32] J.P. Hacker, J. Exby, D. Gill, I. Jimenez, C. Maltzahn, T. See, G. Mullendore, and K. Fossell, "A containerized mesoscale model and analysis toolkit to accelerate classroom learning, collaborative research, and uncertainty quantification," *Bulletin of the American Meteorological Society*, Oct. 2016.

[33] I. Jimenez, "Following Popper for Papers in the Mathematical Sciences," *Popper Wiki*, Dec. 2016. Available at: https://github.com/systemslab/popper/wiki/Popper-Math-Science.

[34] 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.