Reproducibility: Leveraging PopperCI and Toil to Automate the Process of Running Cactus Reproducibly and Conduct a Parameter Sweep Thereof

Bettie D. Jea
Department of Computer Science
University of California*
Santa Cruz, CA, USA
bjea@ucsc.edu

In light of the recent developments in distributed systems and big-data cloud computing, the importance of reproducibility in computer science experiments cannot be stressed enough. In addition to reproducibility, a key to the success of a software package is its user-friendliness—the ease of use and learnability. To save time on the nuances and focus more on the research itself, we want to stand on the shoulders of giants—which is why I introduce PopperCI, Toil, and the combination thereof. PopperCI—the integration of the Popper Convention and existing continuous integration (CI) systems/services—enables researchers to automate end-to-end execution and validation of computer science experiments. Toil is an open-source, portable, and pure-Python workflow software package that supports contemporary workflow definition languages and can be used to securely, reproducibly run scientific workflows efficiently on a grand scale in cloud or high-performance computing environments. This project demonstrates two examples of leveraging PopperCI to fully automate Toil workflows reproducibly. Because of the enormous benefits of PopperCI and Toil, I hope that more and more people will take advantage of them, and that along the way, people will provide the community with helpful feedback to improve their features or even some useful modules to further enrich their feature sets, which is absolutely a virtuous circle.

1 Background to Genomic Data Processing

The cost of sequencing a human genome has decreased dramatically over the past quarter-century, from $100M in 2001 to $1000 in 2015 per genome, which is way below the prediction of $1M by Moore's Law [1]; in addition to the breakthrough achieved by next-generation DNA sequencing (NGS) [2], we must credit most of the success to the great improvement in computing—both in hardware and software—especially *open-source software*.

A genome is composed of all the DNA in a cell's nucleus; DNA consists of four chemically different building blocks, or "bases" (namely, A, T, G, and C, abbreviated for simplicity). The order of these bases determines the biological information encoded within DNA, e.g., the genes determining the structure of proteins. (Not all the DNA sequences are genes, but every single gene is a sequence of DNA.) A gene is a sequence of DNA that occupies a specific location on a chromosome and can be transcribed into a sequence of RNA that may be translated into an amino-acid chain (or a protein) or function directly [3]; a chromosome is a linear strand of DNA plus associated proteins in the nucleus of eukaryotic cells [4].

*An awesome university!

Submitted to:
CMPS 296 Master's Project, Fall 2017

Hence, a change in a DNA sequence may result in a structural change in the corresponding protein. Nearly all the DNA of diploid organisms, e.g. humans (23 pairs of chromosomes) and all other mammals, have duplicate copies because chromosomes are in pairs, with one chromosome of each pair inherited from one parent [5]. Generally, the size of an organism's genome is the total number of bases in one representative copy of its nuclear DNA [5]. In the case of humans (and all other diploid organisms), that corresponds to the sum of the bases of one copy of each chromosome pair; a single 'representative' copy of the human genome is $\sim$3 billion bases in size, whereas a given person's actual (diploid) genome is $\sim$6 billion bases in size [5].

Before sequencing can be performed, a genomic DNA sample has to be broken randomly into smaller sizes from which templates are created; those templates are in turn amplified (usually to 20-50 times) and then sequenced [6]. Therefore, the result of genomic sequencing (or DNA sequencing) is strings of characters (i.e., A, T, G, and C), which are called "reads," and because of the amplification, each fragment of DNA has an average of 20-50 copies [6]. The most significant breakthrough of NGS technologies is the ability to produce a vast quantity of data cheaply—at least relatively cheaply [2]. Depending on the platforms/machines used, the data produced can average around 0.45 to 50 billion bases (gigabases) per run with a run time ranged from 0.35-14 days [6]. The file size of a human genome sample right off the genome sequencer is about 200 GB—thus big data.

2 Introduction

Because of the breakthrough of NGS technologies, contemporary genomic datasets often comprise thousands of samples and petabytes (10^{15} or 2^{50} bytes) of sequencing data [7]. In a genomic data processing pipeline, there can be dozens of individual steps, each of which has its own set of parameters [7]. In consideration of such size and complexity, *reproducibility* and computational resource limitations have become a grave concern within genomics; to address these interrelated issues, *Toil* has been created [7].

Toil, which is an open-source and portable workflow software package that supports contemporary workflow definition languages, enables scientific workflows to run not only securely, efficiently (even on a large scale) but also reproducibly [8]. In light of the recent significant developments in distributed systems and big-data cloud computing, the importance of *reproducibility* in computer science experiments cannot be stressed enough—now more than ever—since reconstructing a similar environment to one where an experiment was originally carried out is a time-consuming endeavour [9]. In addition to reproducibility, a key to the success of a software package is its user-friendliness—the ease of use and learnability.

Over the past decade, there has been a substantial number of bio-, genomics-, transcriptomics-, and proteomics-related packages, libraries developed to save time on the nuances of parsing, handling different data formats, and so on. Among them, the popular ones are usually not only useful but also *easy to use*. Just like programming languages Python and R, they have been dominating the computational biology and bioinformatics fields for years [10], one of the primary reasons being that they are easier to begin with. Therefore, more and more people are willing and able to get involved in big-data processing, who may in turn provide feedback to and even contribute to the community. Because of the feedback and contribution, existing libraries, packages are often improved while new ones can be created, which is likely to attract more people to use the language, and thus more feedback, contribution—a virtuous circle!

Accordingly, usability and user-friendliness are crucial factors in a software package's success. Toil has already enabled and simplified reproducible, rapid large-scale analyses across multiple environments [8], and if the process of running Toil can be even further simplified, it will become even more popular. Furthermore, if the tools, such as Cactus (a reference-free, whole-genome multiple-sequence alignment program), using Toil as their job coordinator can become more user-friendly, say, running with one command across multiple environments, more people would want to use them. Ergo, PopperCI [9] is introduced.

PopperCI, the integration of the Popper Convention [11] and existing continuous integration (CI) systems/services [12], enables researchers to automate end-to-end execution and validation of computer science experiments [9]. PopperCI assumes that experiments conform to Popper, a convention on conducting computational experiments and writing articles adopting a DevOps approach [9, 11, 13]. In this project, I leverage PopperCI to fully automate and further simplify the process of running Cactus (Toil workflows) reproducibly; moreover, an automated parameter sweep is demonstrated.

3 Popper and Experiment Validations

In this section, Popper and one unique component of it, *i.e.* experiment validations, are briefly introduced.

3.1 Popper

Popper puts the idea of an executable paper into practice; it carries out the integration of executables and data for research articles to facilitate its reproducibility [9]. Popper is a convention that systematically implements the different stages of an experimental process following a DevOps approach [9]. To put it in a nutshell, the Popper Convention comprises three high-level guidelines:

1. Pick a DevOps tool for each stage of the scientific experimental workflow;
2. Put all pertinent scripts (experiments and manuscripts) in version control to provide a self-contained repository; and
3. Document changes, in the form of version-control commits, as an experiment evolves [9].

Similar to an open-source software (OSS) project, researchers, by following these guidelines, can make such self-contained repository publicly available with the goal of minimizing the efforts (for others) to perform and validate those experiments again [9].

3.2 Experiment Validations

Although experiment validation is an optional component in Popper, it is useful because these domain-specific tests can confirm that the claims made about the results of an experiment are supported after every re-execution [9]. *Experiment validations* are explicitly codified, e.g., "algorithm X outperforms algorithm Y"; or "the runtime of algorithm Z is at least 8x better than the baseline when the level of parallelism exceeds 2 concurrent threads," and the output of validations includes Boolean values (true/false).

4 PopperCI

PopperCI, the integration of the Popper Convention and existing continuous integration (CI) systems/services [12], automates the validation of the claims made about the results of an experiment after a new commit is pushed to the repository that stores the experiment [9, 12]. As long as a user has created an account at a CI service, e.g. Travis CI [14], and installed a git hook in the local repository, such automation is enabled [9]. PopperCI currently supports Travis CI, and support for Buildbot [15], CircleCI [16], Concourse [17], GitLab CI/CD [18], and Jenkins [19] is on the way. PopperCI enables researchers to automate end-to-end execution and validation of computer science experiments [9].

PopperCI assumes that an experiment is structured to conform to the Popper Convention, which enables the creation of self-contained experiments and articles, and reduces a considerable amount of work that is required to re-execute those experiments [9]. Therefore, to take advantage of the PopperCI service, the folder of an experiment is structured in a specific way (see Figure 1) so that when a new commit is pushed to its associated repository, the service takes the following steps:

1. Ensure that the dependencies in every version are healthy, e.g., make certain that external repositories can be correctly cloned;
2. Check the integrity of each special subfolder (see Section III-B of [9]);
3. Trigger an execution (invoke `run.sh`) for every experiment, and if applicable, launch the experiment on remote infrastructure (see Section III-D of [9]);
4. Execute validations on the output (invoke `validate.sh`) after the experiment finishes; and
5. Keep track of every experiment and report its status [9].

Figure 1: Basic structure of a Popper repository. Source: [9].

```
$> tree -a paper-repo/experiments/myexp
paper-repo/experiments/myexp/
|-- README.md
|-- .popper.yml
|-- run.sh
|-- setup.sh
|-- validate.sh
```

For each *Popperized* project, each experiment (for every commit) has three possible statuses: `FAIL`, `PASS`, or `GOLD` [9]. Each validation has two possible statuses: `FAIL` or `PASS` [9]. If the *experiment status* is `FAIL`, it means that the experiment has failed to execute (`setup.sh` or `run.sh` failed) and thus, validations do not get to be checked at all (`validate.sh` not run) [9]. If the *experiment status* is `PASS`, it means that the experiment runs correctly, but that at least one validation has failed (the status of at least one validation being `FAIL`) [9]. The *experiment status* of `GOLD` means that the status of all validations is `PASS` [9]. PopperCI provides badges that projects can include on its README page on the web interface of the version control system, e.g. GitHub [9]. Badges available for PopperCI are `FAIL`, `OK`, and `GOLD` [9].

5 Toil

Toil is an open-source, portable, and pure-Python workflow software package that supports contemporary workflow definition languages, is designed around the principles of functional programming, and can be used to securely, reproducibly run scientific workflows efficiently on a grand scale in cloud or high-performance computing (HPC) environments [8]; all these features of Toil enable people to write better pipelines.

5.1 Reproducibility

Toil is the first software to not only execute Common Workflow Language (CWL) but also provide draft support for Workflow Description Language (WDL), which makes the sharing of scientific workflows significantly easier because both workflow standards have been thriving [7]. A workflow comprises a set of tasks, or "jobs," that are orchestrated by setting out a set of dependencies that map the inputs and outputs accordingly among those jobs [7]. Toil allows workflows to be declared statically or generated dynamically *via* its Python API (Application Program Interface) so that jobs are capable of defining further jobs during execution if needed [8].

The jobs defined in either Python or CWL may be composed of Docker containers, which enables the sharing of a program without requiring tool installation or configuration in a particular environment [7]. Therefore, containerized workflows are able to be run accurately and reproducibly regardless of the environment [8]. Moreover, Toil supports services, such as databases and servers, that are defined and managed within a workflow, and by this mechanism, it integrates with Apache Spark and is capable of rapidly creating containerized Spark clusters within the context of a larger workflow [8].

5.2 Portability

Toil can run in various cloud environments, including those of Amazon Web Services (AWS), Microsoft Azure, Google Cloud, and OpenStack, as well as in HPC environments that run GridEngine or Slurm and in distributed systems that run Apache Mesos [8]. Toil can be installed with a single command (`pip install toil`) and also runs on a single machine, e.g. a laptop or workstation, to allow for interactive development [7].

Toil's portability is attributed to pluggable back-end APIs for machine provisioning, job scheduling, and file management [8]. Toil manages checkpointing and intermediate files shared among jobs through a "job store," which can be an object store like AWS's S3 (Simple Storage Service), a network file system, or standard file systems [7, 20]. The *job store* interface is an abstraction layer that conceals the specific details of file storage [20]. Thanks to the flexibility of Toil's back-end APIs, we can run a single script on any supported compute environment, coupled with any job store, without modifying the source code [7].

5.3 Scalability and Efficiency

Toil has proved its scalability and efficiency by processing over 20,000 RNA-Seq samples (involving 368,000 jobs) within 90 hours of wall-clock time by using up to 32,000 AWS preemptible cores and 1,325,936 core-hours—more than thirty times faster than TCGA's best-practice workflow [7]. The cost per sample was US$1.30—again, more than thirty times cheaper than the same TCGA workflow—while

achieving a 98% gene-level concordance with said workflow's expression predictions [7].

In addition to Toil's leader/worker pattern for job-scheduling, where a leader makes major decisions on job delegation while workers intelligently decide whether they have the capability, based on resource requirements and workflow dependencies, to run the jobs immediately downstream to their assigned task, Toil implements file caching and streaming [7]. Whenever possible, consecutive jobs sharing files are scheduled on the same node, and file caching obviates the need for transparent transfers from the file store repeatedly [7].

Such low cost also stems from Toil's robustness to job failure—workflows can be resumed after any combination of leader and worker failures—so that Toil workflows can use lower-cost preemptible machines, which may be terminated by the service provider on short notice and are currently available at a substantial discount on AWS and Google [7].

6 Cactus

Cactus is a reference-free, whole-genome multiple-sequence alignment program and uses Toil to coordinate its jobs [21, 22]. Cactus, through Toil, supports many batch systems, including GridEngine, LSF (Platform Load Sharing Facility), Parasol, SLURM, and Torque [21]. Cactus uses a cactus graph (or cactus tree), a connected graph in which any edge is a member of at most one simple cycle, to construct a genome alignment; a simple cycle is one in which no edge or vertex is repeated except for the start/end vertex [22, 23]. The aligner permits arbitrary types of duplication and rearrangement, but does favor alignments that create "chains" of aligned bases [22, 24].

7 Reproducibility: Leveraging PopperCI and Toil to Automate the Process of Running Cactus Reproducibly

Toil has provided us with a portable, scalable, and reproducible workflow software package that is open-source and can be run in cloud or HPC environments. Toil's user-friendliness and detailed documentation have made it easy to run scientific workflows on a large scale, and if the process can be further simplified and fully automated, it will be even more attractive to the general public. Although Toil is user-friendly and well-documented, some cloud service providers may not be.

However, a lot of big-data processing that is highly parallelizable prefers to be run in the cloud to save time. Furthermore, many biologists who are not computer science majors or bioinformaticians also would like to run biocomputing programs, but are concerned about the overhead of getting familiar with setting up the environment, running the program, etc. and the likely overhead for the teammates. If we could further simplify the process so that users only need to clone a self-contained repository, change some parameters of interest, and provide their own datasets, more and more people would find it attractive to use Toil to run scientific workflows on a huge scale.

Most of those very popular packages, libraries are popular for a reason—they are not only useful but also *easy to use*. In addition to the user-friendliness, reproducibility is even more crucial to the scientists because if an experiment cannot be repeated by an author or replicated by others using the original arti-

facts, or not be reproduced by anyone re-implementing the experiment, then it is *not* scientifically sound.

To achieve user-friendliness, full automation, and reproducibility, I hereby leverage Popper and Toil to completely automate the process of running Cactus program in a reproducible way (regardless of the environments and configurations) and conducting a parameter sweep thereof. This is just an example to show that by following the Popper Convention, researchers are capable of automating the process of using Toil software to run scientific workflows, that by making use of PopperCI, researchers are able to ensure that their work is reproducible, which makes it easier to share and collaborate with others [9].

7.1 Version Control

Following the Popper Convention, the content managed in a version-control system (VCS) includes: (a) for a project, 1) source code, 2) data and figures, if applicable; (b) for an academic article, 1) article text, 2) experiments (code and data), and 3) figures [11].

Tools and Services: Git and GitHub. Git is one of the most popular VCS tools; GitHub is a web-based Git repository hosting service. Because Toil project uses Git and GitHub as its VCS tool and service, and I personally also prefer them, they are selected for version control in this experiment. Git is able to keep track of changes in any files and is targeted at data integrity, speed, and support for distributed, non-linear workflows [25]. Unlike most client-server systems, every Git directory (including those cloned from a repository) on every computer is a full-fledged repository with the complete history and its artifacts and full version-tracking abilities, independent of a central server or network access [25]. GitHub offers source code management, all the features of Git distributed VCS as well as adding features like bug tracking, feature requests, and wikis for every repository [26].

7.2 Package Management

Even though the source code is available, an experiment is *not* guaranteed to be reproducible because of the dependency requirements and possible differences in environments and/or operating systems. Therefore, packaging of applications allows users to *not* have to re-create it by themselves so that it fulfills the purpose of toolchain agnosticism and easy deployment across environments [11]. Software containerization (operating-system-level virtualization [27]), such as Docker, FreeBSD jails, LXC, and OpenVZ, completes package managers by packaging all the dependencies of an application in a snapshot (e.g. a Docker image) that is able to be deployed in systems "as is" without worrying about problems like package dependencies or specific OS versions because the snapshot contains a union of layered filesystems stacked on top of one another [11, 28].

Tools and Services: Docker. Toil project uses Docker as its package management tool, and I personally also prefer using it. Docker provides an additional layer of abstraction and automation of containerization on Linux and Windows, and thus, Docker can automate the deployment of applications inside software containers [11].

7.3 Validation

The output file of Cactus is a genome-sequence alignment in the HAL format, which represents the alignment in a reference-free, indexed way [21]. In more detail, HAL format is a compressed, graph-based

hierarchical alignment format for storing multiple genome alignments and ancestral reconstructions [29]. However, comparing two output HAL files is not practical for two prongs: (a) on one level, there is going to be some minute amount of noise in the alignment results even between identical runs, which is due to the differences in the ordering of what jobs get run when; and (b) on the HDF5 [30] (HAL being one of the HDF5 file formats) level, there are bound to be tons of differences because even for the exact same alignment, many different HDF5 representations are possible (differing solely in the ordering of sequences within a genome or the like) [31]. As a result, it may not make too much sense to put effort into consistently generating the same representation [31].

To compare the alignments, the `hal2maf` tool, which converts HAL to MAF, is used to export the alignment from any particular genome [21], and then the mafComparator from the mafTools suite is usually employed to compare the alignment files on the alignment level, to examine how many pairs in sequence alignment are shared between two files [31]. Thus, I created a Docker image, bjea/maftools, to run the `mafTools` suite.

However, I have found that even with two identical runs (identical values for all parameters), the resulted MAF files may still have slight difference (usually in the length of sequence of just one chromosome). This difference is likely to stem from the uncertainty in the sequence alignment [32] and inherent statistical limitations when it comes to the accuracy of the alignments produced by any algorithms [33]. The uncertainty may arise from multi-dimensional discrete spaces, in which the number of possible solutions is enormous [34]. Based on other people's research and studies [24, 32, 33], such discrepancy relates to alignment uncertainty rather than the irreproducibility of the experiment procedure.

Because to fight against the uncertainty is beyond the scope of this project and my ability, I would be content with procedural reproducibility, instead of outcome-based reproducibility, for now. Consequently, for the validation part of the Cactus program, since I cannot combat the uncertainty and do not have a 'golden file' as the expected output of a test, I simply run a unit test, using the genome sequences of four biovars of *Y. pestis* [35, 36], to see if it can be completed.

7.4 Continuous Integration

CI is especially important when we are contributing our code to a shared repository; we want to catch errors as early as possible.

Tools and Services: PopperCI. PopperCI—the integration of the Popper Convention and existing CI systems/services [12]—automates the validation of the claims made about the results of an experiment after a new commit is pushed to the repository that stores the experiment [9, 12]. As long as a user has created an account at a CI service, e.g. Travis CI [14], and installed a git hook in the local repository, such automation is enabled [9]. PopperCI currently supports Travis CI, and will provide support for Buildbot [15], CircleCI [16], Concourse [17], GitLab CI/CD [18], and Jenkins [19] in the near future.

7.5 Execution

First, I have started with a sorting pipeline [37] that merge-sorts a file using a Toil workflow to show that following the Popper Convention allows a user to fully automate and further simplify the process of running a Toil workflow; moreover, an automated parameter sweep is demonstrated. Second, I run

Cactus, which uses Toil to coordinate its jobs, conforming to the Popper Convention and using Travis CI as the CI tool.

7.5.1 MergeSort Pipeline

This merge sort is *smart* because each step of the process—dividing the file of interest into separate chunks, sorting these chunks, and merging these sorted chunks back together—is compartmentalized into a "job" of a Toil workflow [37]. Each "job" is able to specify its own resource requirements and will be run only after the jobs that it depends upon have run [37]. Jobs without dependencies can be run in parallel [37].

Following the Popper Convention, I am able to fully automate and further simplify the process of running this mergesort workflow on a local machine or in cloud (e.g. AWS) and conduct a parameter sweep thereof. Now a user needs to type in merely one command—`popper check`—of the Popper CLI tool[1] (*cf.* 14 commands using a traditional method, see Table 1), and everything, including the parameter sweep if desired, works like a charm—exactly as expected.

In this case, the parameters used, e.g. `--numLines` (number of lines), `--lineLength` (how many characters in each line), are not for optimization purposes since this sorting program is designed not to run fast but rather to showcase a Toil workflow and its capability—also to showcase Popper's user-friendliness and convenience. More importantly, now that the `sort` example is a self-contained, Popper-compliant experiment, its reproducibility is warranted.

7.5.2 Cactus Program

Cactus, which uses Toil to coordinate its jobs, is a reference-free, whole-genome multiple-sequence alignment program. Following the Popper Convention, I am able to completely automate and further simplify the process of running Cactus in cloud (e.g. AWS) and conduct a parameter sweep thereof. Because Cactus will take about 20 CPU-hours per bacterium-sized (∼4 megabases) genome, about 20 CPU-days per nematode-sized (∼100 megabases) genome, and about 120 CPU-days per mammal-sized (∼3 gigabases) genome, it may not be practical to run on my machine.

Therefore, I only ran Cactus in AWS; it should, nevertheless, work fine on a single machine—just take more time. One of the organisms I used that is also used as a unit-test genome sequence for CI is *Yersinia pestis* [35, 36] (∼4.6 megabases), which is a Gram-negative, rod-shaped coccobacillus, a facultative anaerobic organism that can infect humans via the oriental rat flea and can cause the deadly disease called Bubonic Plague (or "the Plague") [38].

When we run Cactus in AWS, it is running inside a Docker container—a runtime instance of the Docker image quay.io/ucsc_cgl/toil—which contains the Mesos scheduler and other requirements. Then, the Cactus script gets installed in a virtual environment (or `virtualenv`) within such container, and a number of other containers (with the same quay.io/ucsc_cgl/toil image) are spun up on worker nodes

[1] If on a local machine, one command needed: `popper check`; if in cloud, two commands required: (1) `bash setup.sh`, and (2) `popper check --skip setup.sh`, which is because the value of flag `--mesosMaster <private-IP>:5050` cannot be obtained till the leader node is launched, the `<private-IP>` needs to be filled in one of the files uploaded to the leader node.

launched by the leader node. The virtual environment is faithfully copied over to all the workers so that they all have a copy of the pipeline orchestrated by Toil. To achieve the purpose of easy deployment across environments, I would like to run every single step of the experiment in a Docker container; however, it may not be realistic in this case. Because a Cactus workflow is already running in containers, somehow Docker within Docker did not work after many attempts. It may be possible to create a specialized image for a particular Toil workflow so that a virtual environment is not necessary, and that the Toil workflow can still be distributed/copied to the worker nodes. I, nevertheless, was not able to succeed.

Accordingly, when it is on a local machine, every single step runs in a Docker container. While it is on a remote leader node in cloud, the experiment is run in a virtual environment within a container after dependencies are installed. This does not defeat the purpose of toolchain agnosticism or easy deployment across environments even though I am not able to use another container to run the already-containerized Cactus workflow. Because if we want to use Toil to run our pipeline in cloud, a leader node of CoreOS Container Linux [39] OS is launched (using the command `toil launch-cluster <cluster-name>`) so that we need not worry about a virtual environment or library dependencies being not able to be activated or installed.

Now a user can type in merely one command—`popper check`—of the Popper CLI tool (*cf.* 31 commands[2], see Table 1), and everything, including the parameter sweep if desired, works like a charm—exactly as expected. In this case, the parameters used include `--configFile` (a Cactus parameter); `--minNodes`, `--maxNodes`, and `--nodeTypes` (Toil parameters, which should not change the output results). I did not check which configuration file yields the best result compared with the true alignment of the sequence since it is not the purpose of this project.

However, by observing the runtime of my experiment during the parameter sweep, I have found that one of the more cost-efficient ways (based on the AWS pricing of on-demand, non-preemptible instances/nodes) to align multiple 4-megabase sequences using Cactus in AWS cloud (if the Amazon spot market [41] not considered) is to use `c4.4xlarge` instance (slightly cheaper than `c3.4xlarge`, but similarly efficient) as a worker-node type (i.e., `--nodeTypes c4.4xlarge`). Or it can use smaller instances, e.g., `c4.2xlarge`, `c4.xlarge`, since Toil is so *smart* that if the resources are not enough with one worker node, it will launch more (just set `--maxNodes` to a higher value), and AWS's pricing is usually proportional to the compute capacity, e.g. `c4.4xlarge`'s cost being the double of `c4.2xlarge`'s and the quadruple of `c4.xlarge`'s [42]. The cost of on-demand, non-preemptible instances is generally calculated by the hour (if less than an hour, counted as an hour). If there are many samples to run and time is not really of the essence, running a small-scale parameter sweep to find the optimal worker-node type to utilize can be cost-effective. For example, if it is not urgent, using `c4.8xlarge` would be overkill since a `c4.4xlarge` worker node can finish four 4-megabase-sequence aligning within 45 minutes.

Following the Popper Convention ensures that an experiment is not only procedurally reproducible but also easy to carry out. Users can use a *Popperized* experiment as an experiment package so that it can be easily replicated and that they can easily change the datasets and parameters of interest to conduct their own experiments.

[2] If a traditional method is used, and it starts from preparing for and installing the Toil software [40], but does not take encryption into consideration, and if only four genomic sequences are used as samples to run Cactus.

Experiment	**Traditional Method (No. of Commands Needed)**	**Popper Method (No. of Commands Needed)**
MergeSort*	14	2**
Cactus Program***	31	1

Table 1: Comparison of Number of Commands Needed: Traditional vs. Popper.
* For traditional method, it starts from preparing for and installing the Toil software [40], but does not take encryption into consideration.
** If on a local machine, one command needed: `popper check`; if in cloud, two commands required: (1) `bash setup.sh`, and (2) `popper check --skip setup.sh`, which is because the value of flag `--mesosMaster <private-IP>:5050` cannot be obtained till the leader node is launched, the `<private-IP>` needs to be filled in one of the files uploaded to the leader node.
*** For traditional method, it starts from preparing for and installing the Toil software [40], but does not take encryption into consideration, and only four genomic sequences are used as samples to run Cactus.

8 Conclusion

To save time on the nuances and focus more on the research itself, we want to stand on the shoulders of giants—which is why I introduce PopperCI, Toil, and the combination thereof. Being fast and handy is important to bio folks (and many others); here, being *fast* is more about saving the "programmer time" [43], instead of CPU time. Ergo, being easier to use/debug, having faster code-test-deploy cycles, having a lot of features, community support, and online information (e.g. well-delineated documentation) have saved developers significant amount of time on tedious tasks so that they can spend more time on the research itself.

At the beginning, it may seem a little bit burdensome to follow the Popper Convention since programmers tend to have their own ways of running an experiment, but after a while, the enormous benefits of being *Popperized* will be appreciated, and Popperizing an experiment just becomes natural and effortless! In the long run, programmers would be so glad that they Popperized their experiments because who would remember what s/he wrote or how s/he set up an environment three months ago? Following Popper enables us to run an experiment automatically—with merely *one* command—as well as reproducibly since every experiment is a self-contained repository. Similar to those naming conventions, conforming to them can be a pain at the start, but soon you would realize how helpful it is.

Likewise, using Toil to orchestrate jobs in a pipeline may seem a bit complicated at first; however, it is totally worth it in the long run. Toil allows secure, efficient, and reproducible big-data analyses to be performed easily; it just has a steep learning curve. More importantly, Toil and all the tools mentioned by PopperCI are all *open-source software*, so if anyone wants to reproduce a Popperized experiment using Toil, s/he can do so without trouble.

This project simply demonstrates two examples of leveraging PopperCI to fully automate Toil workflows reproducibly. Because of the great benefits of PopperCI, Toil, and Cactus, I hope that more and more people will take advantage of them, and that along the way, people will provide the community with helpful feedback to improve their features or even some useful modules to further enrich their feature sets, which is absolutely a virtuous circle!

Acknowledgments

First and foremost, I would like to express my warmest thanks to my inspiring advisor, Dr. Carlos Maltzahn, for all he has done for me. He has always been there to provide guidance whenever needed. I am truly grateful for his encouragement, support, and patience, which mean a lot to me. It has been a great privilege to be under the tutelage of such an admirable, brilliant, and knowledgeable scientist of profound wisdom.

Moreover, a B-I-G, grateful thanks to Joel Armstrong, who is one of the main contributors to Cactus, for his M-O-S-T helpful and detailed responses to my questions, which truly cleared up my confusion.

I would also like to express my heartfelt thanks to Dr. Edwin Jacox, who is the leader of the Toil Dev team, for his wonderfully generous help and eminently valuable advice. He was always willing to help me, discuss with me, and solve my problems. He was so kind that he took me under his wing and let me work on my project on his team. In addition, I want to thank his teammates Lon Blauvelt and Jesse Brennan, who are nice, friendly, and helpful.

Furthermore, I would like to thank Ivo Jimenez, who is the main developer of Popper and PopperCI, for his VERY kind help and guidance. I deeply appreciate his prompt replies to my emails, his willingness to spend time teaching me, and his positive, cheerful attitude; additionally, he is passionate about his research. Every time I get his approval, I feel really happy and totally encouraged!

Further, I would like to express my sincere gratitude to the members of my committee (alphabetized by last name), Dr. Peter Alvaro, Dr. Benedict Paten, who is also the director of the Computational Genomics Lab of Genomics Institute, for contributing their precious time and energy. Both of them are super busy but still agree to serve on my committee, of which I am deeply appreciative.

Last but not least, I want to profusely thank my parents for their wholehearted support and considerable encouragement throughout my life. They unswervingly believe in me. Without them, I am nothing.

References

[1] Kris A. Wetterstrand (May 24, 2016): *DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP)*. Available at https://www.genome.gov/sequencingcostsdata/. Accessed 12/06/2017.

[2] Stephan C. Schuster (2008): *Next-generation sequencing transforms today's biology*. Nature Methods 5(1), pp. 16–18. Available at http://search.proquest.com/openview/6356e1eab10aa6ea294804d784ec994f/1?pq-origsite=gscholar&cbl=28015.

[3] American Heritage® Dictionary of the English Language (Fifth Edition) (2011): *Gene*. Available at http://www.thefreedictionary.com/gene. Accessed 12/06/2017.

[4] American Heritage® Dictionary of the English Language (Fifth Edition) (2011): *Chromosome*. Available at http://www.thefreedictionary.com/chromosome. Accessed 12/06/2017.

[5] Large-Scale Genome Sequencing & Analysis Centers (LSAC) of NIH (July 6, 2016): *The Cost of Sequencing a Human Genome*. Available at https://www.genome.gov/sequencingcosts/. Accessed 12/06/2017.

[6] Michael L. Metzker (2010): *Sequencing technologies—the next generation*. Nature Reviews Genetics 11, pp. 31–46, doi:10.1038/nrg2626. Available at http://www.nature.com/nrg/journal/v11/n1/pdf/nrg2626.pdf.

[7] John Vivian, Arjun Rao, Frank Austin Nothaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D. Deran, Audrey Musselman-Brown, Hannes Schmidt, Peter Amstutz, Brian Craft, Mary Goldman, Kate Rosenbloom, Melissa Cline, Brian O'Connor, Megan Hanna, Chet Birger, W. James Kent, David A. Patterson, Anthony D. Joseph, Jingchun Zhu, Sasha Zaranek, Gad Getz, David Haussler & Benedict Paten (2016): *Rapid and efficient analysis of 20,000 RNA-seq samples with Toil*. *bioRxiv*, doi:https://doi.org/10.1101/062497. Available at https://www.biorxiv.org/content/early/2016/07/07/062497.

[8] John Vivian, Arjun Rao, Frank Austin Nothaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D. Deran, Audrey Musselman-Brown, Hannes Schmidt, Peter Amstutz, Brian Craft, Mary Goldman, Kate Rosenbloom, Melissa Cline, Brian O'Connor, Megan Hanna, Chet Birger, W. James Kent, David A. Patterson, Anthony D. Joseph, Jingchun Zhu, Sasha Zaranek, Gad Getz, David Haussler & Benedict Paten (2017): *Toil enables reproducible, open source, big biomedical data analyses*. *Nature Biotechnology* 35, pp. 314–316, doi:10.1038/nbt.3772. Available at https://www.nature.com/articles/nbt.3772.

[9] Ivo Jimenez, Sina Hamedian, Carlos Maltzahn, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Kathryn Mohror, Jay Lofstead & Robert Ricci (2017): *PopperCI: Automated Reproducibility Validation*. In: *IEEE International Conference on Computer Communications (INFOCOM); Proceedings of the International Workshop on Computer and Networking Experimental Research Using Testbeds (CNERT)*. Available at http://falsifiable.us/pubs/.

[10] Bohdan B. Khomtchouk, Edmund Weitz, Peter D. Karp & Claes Wahlestedt (2016): *How the strengths of Lisp-family languages facilitate building complex and flexible bioinformatics applications*. *Briefings in Bioinformatics* bbw130, doi:10.1093/bib/bbw130. Available at https://academic.oup.com/bib/article-lookup/doi/10.1093/bib/bbw130.

[11] Ivo Jimenez, Michael Sevilla, Noah Watkins, Carlos Maltzahn, Jay Lofstead, Kathryn Mohror, Remzi Arpaci-Dusseau & Andrea Arpaci-Dusseau (2016): *Popper: Making Reproducible Systems Performance Evaluation Practical*. Technical Report, UCSC-SOE-16-10, Department of Computer Science, School of Engineering, University of California Santa Cruz, Santa Cruz, CA 95064. Available at https://www.soe.ucsc.edu/sites/default/files/technical-reports/UCSC-SOE-16-10.pdf.

[12] Ivo Jimenez (2017): *Popper — Getting Started — Adding Project to Travis*. Available at http://popper.readthedocs.io/en/latest/protocol/getting_started.html#adding-project-to-travis. Accessed 12/05/2017.

[13] Ivo Jimenez, Michael Sevilla, Noah Watkins, Carlos Maltzahn, Jay Lofstead, Kathryn Mohror, Andrea Arpaci-Dusseau & Remzi Arpaci-Dusseau (2016): *Standing on the Shoulders of Giants by Managing Scientific Experiments Like Software*. *USENIX ;login:* 41(4), pp. 20–26. Available at https://www.usenix.org/publications/login/winter2016/jimenez.

[14] Travis CI, GmbH (2017): *Travis CI*. Available at https://travis-ci.org/. Accessed 12/01/2017.

[15] Buildbot — originally written by Brian Warner & now maintained by Dustin J. Mitchell (2017): *Buildbot*. Available at https://buildbot.net/. Accessed 12/06/2017.

[16] CircleCI (2017): *CircleCI*. Available at https://circleci.com/. Accessed 12/06/2017.

[17] Concourse, Sponsored by Pivotal Software, Inc. (2017): *Concourse*. Available at https://concourse.ci/. Accessed 12/06/2017.

[18] GitLab (2017): *GitLab Continuous Integration & Deployment*. Available at https://about.gitlab.com/features/gitlab-ci-cd/. Accessed 12/06/2017.

[19] Jenkins Open-Source Software Project (2017): *Jenkins*. Available at https://jenkins.io/. Accessed 12/01/2017.

[20] Computational Genomics Lab, Genomics Institute, University of California Santa Cruz (2017): *Toil - Job Store API*. Available at https://toil.readthedocs.io/en/3.12.0/developingWorkflows/jobStore.html. Accessed 11/22/2017.

[21] Computational Genomics Lab, Genomics Institute, University of California Santa Cruz (2017): *GitHub - ComparativeGenomicsToolkit/cactus*. Available at https://github.com/ComparativeGenomicsToolkit/cactus. Accessed 11/24/2017.

[22] Benedict Paten (2017): *Cactus*. Available at http://hgwdev.cse.ucsc.edu/~benedict/code/Cactus.html. Accessed 11/24/2017.

[23] Benedict Paten, Mark Diekhans, Dent Earl, John St. John, Jian Ma, Bernard Suh & David Haussler (2011): *Cactus Graphs for Genome Comparisons*. *Journal of Computational Biology* 18(3), pp. 469–481. Available at http://online.liebertpub.com/doi/abs/10.1089/cmb.2010.0252.

[24] Benedict Paten, Dent Earl, Ngan Nguyen, Mark Diekhans, Daniel Zerbino & David Haussler (2011): *Cactus: algorithms for genome multiple sequence alignment*. *Genome Res.* 21(9), pp. 1512–1528. Available at http://hgwdev.cse.ucsc.edu/~benedict/code/Cactus_files/cactusAlignment.pdf.

[25] Wikipedia (2017): *Git — Wikipedia, The Free Encyclopedia*. Available at https://encyclopedia.thefreedictionary.com/git. Accessed 11/28/2017.

[26] Computer Desktop Encyclopedia (1981-2015): *GitHub*. Available at https://encyclopedia2.thefreedictionary.com/Github. Accessed 11/28/2017.

[27] Wikipedia (2017): *Operating-system-level virtualization — Wikipedia, The Free Encyclopedia*. Available at https://en.wikipedia.org/wiki/Operating-system-level_virtualization#frb-inline. Accessed 11/29/2017.

[28] Docker, Inc. (2017): *Definition of: image — Docker glossary*. Available at https://docs.docker.com/glossary/?term=image. Accessed 11/30/2017.

[29] Glenn Hickey, Benedict Paten, Dent Earl, Daniel Zerbino & David Haussler (2013): *HAL: a hierarchical format for storing and analyzing multiple genome alignments*. *Bioinformatics* 29(10), pp. 1341–1342, doi:10.1093/bioinformatics/btt128. Available at https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3654707/.

[30] The HDF5 Group (2017): *WELCOME TO THE HDF5 SUPPORT PAGE!* Available at https://support.hdfgroup.org/HDF5/. Accessed 12/07/2017.

[31] Joel Armstrong (joelarmstrong) (2017): *ComparativeGenomicsToolkit/cactus > Issues > Cactus Output Files #22*. Available at https://github.com/ComparativeGenomicsToolkit/cactus/issues/22. Accessed 12/03/2017.

[32] Karen M. Wong, Marc A. Suchard & John P. Huelsenbeckr (2008): *Alignment Uncertainty and Genomic Analysis*. *Science* 319(5862), pp. 473–476, doi:10.1126/science.1151532. Available at http://science.sciencemag.org/content/sci/319/5862/473.full.pdf.

[33] Ian Holmes & Richard Durbin (1998): *Dynamic programming alignment accuracy*. *Journal of Computational Biology* 5(3), pp. 493–504. Available at https://www.ncbi.nlm.nih.gov/pubmed/9773345.

[34] Michiaki Hamada (2014): *Fighting against uncertainty: an essential issue in bioinformatics*. *Briefings in Bioinformatics* 15(5), pp. 748–767, doi:https://doi.org/10.1093/bib/bbt038. Available at https://academic.oup.com/bib/article/15/5/748/2422287.

[35] Giovanna Morelli, Yajun Song, Camila J. Mazzoni, Mark Eppinger, Philippe Roumagnac, David M. Wagner, Mirjam Feldkamp, Barica Kusecek, Amy J. Vogler, Yanjun Li, Yujun Cui, Nicholas R. Thomson, Thibaut Jombart, Raphael Leblois, Peter Lichtner, Lila Rahalison, Jeannine M. Petersen, Francois Balloux, Paul Keim, Thierry Wirth, Jacques Ravel, Ruifu Yang, Elisabeth Carniel & Mark Achtman (2010): *Phylogenetic diversity and historical patterns of pandemic spread of Yersinia pestis*. *Nature Genetics* 42(12), pp. 1140–1143, doi:http://doi.org/10.1038/ng.705. Available at https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2999892/.

[36] Wilson Sung (2011): *A Thesis: Assessment of orthology identification approaches and the impact of gene fusion and fission in bacteria*. McMaster University, Hamilton, Ontario. Available at https://macsphere.mcmaster.ca/bitstream/11375/9810/1/fulltext.pdf.

[37] Computational Genomics Lab, Genomics Institute, University of California Santa Cruz (2017): *Toil — Quickstart Examples — A (more) real-world example*. Available at https://toil.readthedocs.io/en/3.12.0/gettingStarted/quickStart.html#a-more-real-world-example. Accessed 12/01/2017.

[38] Wikipedia (2014): *Yersinia pestis — Wikipedia, The Free Encyclopedia*. Available at https://encyclopedia.thefreedictionary.com/Yersinia+pestis. Accessed 12/02/2017.

[39] CoreOS, Inc. (2017): *CoreOS*. Available at https://coreos.com/. Accessed 12/03/2017.

[40] Computational Genomics Lab, Genomics Institute, University of California Santa Cruz (2017): *Toil — Getting Started — Installation*. Available at https://toil.readthedocs.io/en/3.12.0/gettingStarted/install.html. Accessed 12/14/2017.

[41] Amazon Web Services, Inc. (2017): *Amazon EC2 Spot Instances*. Available at https://aws.amazon.com/ec2/spot/. Accessed 12/06/2017.

[42] Amazon Web Services, Inc. (2017): *Amazon EC2 Pricing*. Available at https://aws.amazon.com/ec2/pricing/on-demand/. Accessed 12/06/2017.

[43] Erann Gat (2000): *Lisp as an Alternative to Java*. *Intelligence: New Visions of AI in Practice* 11(4), pp. 21–24. Available at http://www.flownet.com/gat/papers/lisp-java.pdf.