Packing Experiments for Sharing and Publication

Fernando Chirigati
Polytechnic Institute of NYU
fchirigati@nyu.edu

Dennis Shasha
New York University
shasha@cs.nyu.edu

Juliana Freire
Polytechnic Institute of NYU
juliana.freire@nyu.edu

ABSTRACT

Reproducibility is a core component of the scientific process. Revisiting and reusing past results allow science to move forward – "standing on the shoulders of giants", as Newton once said. An impediment to the adoption of computational reproducibility is that authors find it difficult to generate a compendium that encompasses all the required components to correctly reproduce their experiments. Even when a compendium is available, reviewers and readers may have difficulties in verifying the results on platforms different from the ones where the experiments were originally run. As a step towards simplifying the process of creating reproducible experiments, we have developed `ReproZip`, a tool that automatically captures the provenance of experiments and packs all the necessary files, library dependencies and variables to reproduce the results. Reviewers can then unpack and run the experiments without having to install any additional software. We will demonstrate real use cases for `ReproZip`, how packages are created, and how reviewers can validate and explore experiments.

Categories and Subject Descriptors

H.2 [**Database Management**]: Database Applications; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Documentation, Experimentation

Keywords

Computational Reproducibility, Provenance, ReproZip

1. INTRODUCTION

The ability to reproduce and test experiments is critical in the scientific method [2, 5], both to verify results and to build on them. In natural science, long tradition requires experiments to be described in enough detail so that they can be reproduced by researchers around the world. This standard, however, has not been applied to computational experiments. Researchers often have to rely on tables, plots and figure captions included in these papers, which loosely describe the obtained results. Since details of the computational steps are often omitted, it is difficult to verify and reproduce many of the published results [9]. This has led to a credibility crisis in computational science [3].

To make a computational experiment reproducible, authors need to encapsulate all the necessary components so that results presented in papers can be verified. A computational experiment, that has been developed at time t on hardware/operating system s and data d, is reproducible if it can be executed at time t' on system s' and data d' that is similar to (or potentially the same as) d. Such experiments are the basic building block for reproducible research papers, which, in addition to text, include data, specification of computational processes, software/code, as well as information about the environment used to derive the results.

In the scientific community, a number of tools have been proposed to support the creation of reproducible experiments. Some of these solutions are domain-specific. For example, GenePattern [6] is a genomic analysis platform, while Madagascar [10] supports multidimensional data analysis and is used to analyze seismic data. Scientific workflow systems, on the other hand, are general and support the specification of arbitrary computational experiments that weave together multiple functions and libraries. While these systems maintain provenance information for workflow executions and data products derived by workflows [1], they fail to capture information about the environment, including software and data dependencies. Thus, even though they support reproducibility, they do not support *portability*: a given workflow may not run in an environment different from the one in which it was originally created.

Another class of tools focuses on capturing information about the computational environment. Examples include virtual machines and CDE [7]. By creating virtual machine snapshots, authors can encapsulate all the components of an experiment. However, such snapshots are often large and encompass not only the components required to reproduce the results, but also a plethora of files that are not related to the experiment. In addition, if authors do not use a virtual machine from the beginning of a project, they will need to create a virtual machine and install the experiment and all dependencies, which can be time consuming. CDE offers a lighter-weight alternative to virtual machines. It relies on the *ptrace* call on Linux to identify only the files required

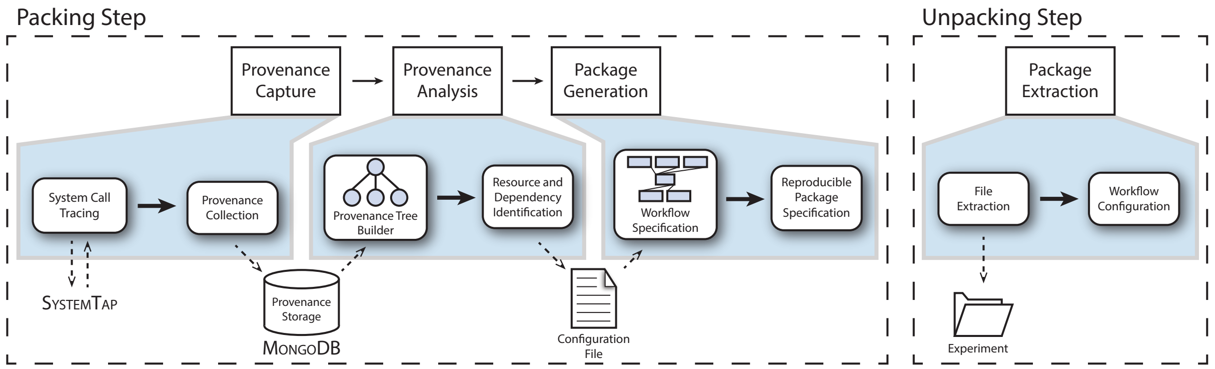

Figure 1: Architecture of `ReproZip`.

for running a particular command, and creates a package containing these files [7]. This package can then be copied to different Linux installations where it can be run within the CDE environment: CDE dynamically changes the system calls to point to the correct files included in the package.

This demo presents `ReproZip`, a general tool that simplifies the process of creating reproducible experiments from command-line executions, a frequently-used common denominator in computational science. Similar to CDE, `ReproZip` tracks operating system calls and creates a package that contains all the binaries, files and dependencies required to run a given command on the author's computational environment E. Unlike CDE, `ReproZip` also generates a workflow specification for the experiment, which can be used to help reviewers to explore and verify the experiment. A reviewer can extract the files and workflow on another environment E' (e.g., the reviewer's desktop), without interfering with any program or dependency already installed on E'. The experiments can then be correctly reproduced and even varied on E'. By using the derived workflow to perform this exploration, provenance of the review process is automatically captured, and can serve not only to document the process but also as a means to support communication between authors and reviewers. Furthermore, users are able to customize the reproducible package and tune is size by interactively inspecting the experiment trace. Last, but not least, `ReproZip` does not add any run-time overhead to executing a packaged experiment. `ReproZip` has been developed to work on Linux distributions, and it has been successfully tested on Ubuntu and Fedora.

2. REPROZIP IN ACTION

`ReproZip` works in two stages: packing and unpacking. Authors use the system to identify the necessary dependencies and to create the reproducible package. The experiment can then be unpacked in the reviewer's environment, where the results are validated and explored.

2.1 Packing Step

The first phase in producing a reproducible experiment on environment E is to *pack* it, which is accomplished by three modules (shown in Figure 1): Provenance Capture, Provenance Analysis, and Package Generation.

Provenance Capture. To create a reproducible experiment, an author invokes his experiment through `ReproZip`. The system uses SystemTap [12] to transparently capture the provenance of the experiment—it dynamically instruments and traces system calls (*execve*, *open*, *read*, *write*, *close* and *pipe*, to name a few). Through these system calls, it is possible to gather information for each computational process involved in the execution of the experiment, such as *command-line arguments*, *environment variables*, *working directory*, *files read* and *files written*. This information is then collected and stored in MongoDB [11], a NoSQL database, where it can be easily accessed and queried. Our choice of SystemTap and MongoDB was inspired by the Burrito System [8], which successfully uses these tools to capture and store provenance for programs run on Linux.

Provenance Analysis. `ReproZip` uses the collected trace data to identify the components of the experiment. The Provenance Analysis module accesses the provenance storage and creates a *provenance tree* of the experiment. Each node in the tree corresponds to an OS process, and an edge between a parent and a child node indicates that the parent process invoked (or spawned) the child process. The root of the tree represents the main process of the experiment which is specified by the user when `ReproZip` is invoked. The tree is built incrementally: when a process corresponding to a node n spawns a process n', a new node is created for n' and an edge is inserted between n and n'. Each node in the tree stores provenance data – obtained by the Provenance Capture module – for the corresponding process.

Once the provenance tree is built, the Resource and Dependency Identification sub-module traverses the tree to identify *executable programs*, *input files*, *output files* and *dependencies* that should be included in the reproducible package. We should note that SystemTap captures *all* dependencies, some of which may not be necessary. `ReproZip` outputs a configuration file that lists all the identified programs, input files and dependencies, and allows authors to *customize* the configuration to exclude a specific file or set of files (e.g., using Unix-shell style wildcards). This customization step is particularly useful to control the size of the package, for example, by discarding temporary files and omitting large files that can be obtained elsewhere.

Package Generation. The identified input and output files are used to derive a specification of the experiment workflow. The main program of the experiment is wrapped in a workflow module that automatically takes the command-line arguments as inputs. By making these arguments explicit in the workflow specification, reviewers can immediately see which parameters can be changed. The current implementation of `ReproZip` derives workflows that can be

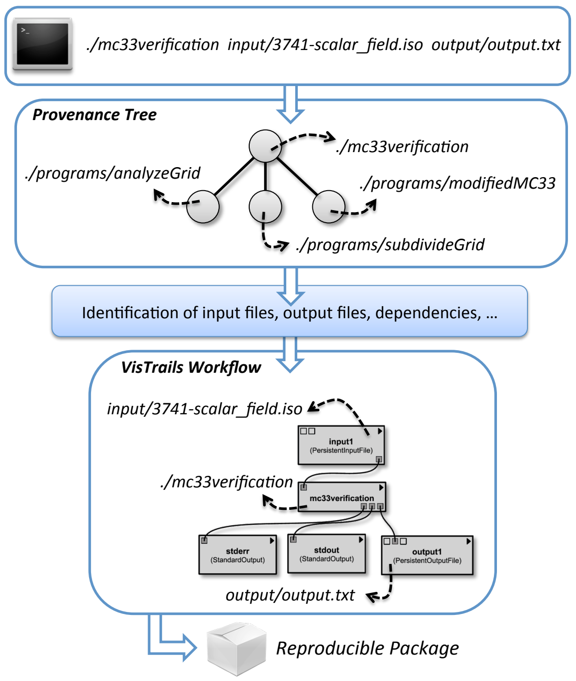

Figure 2: Packing step of an experiment on topological correctness of marching cubes.

run on the VisTrails system (`www.vistrails.org`) [4]. VisTrails supports data exploration and the ability to capture detailed provenance of the review process.

A package is then created that contains the experiment workflow as well as all the required files from the author's environment E, using the same directory structure. Note that the command-line arguments and the environment variables in the workflow are configured to reference the files that are inside the reproducible package. A mapping between symbolic links and target files is also added to the package, so that these links can be correctly created in the unpacking step.

Figure 2 shows a real example where a reproducible package is created for an experiment that verifies the topological correctness of marching cubes algorithms. As shown in the tree, the main program, *mc33verification*, calls three programs: *analyzeGrid*, *subdivideGrid* and *modifiedMC33*. The provenance information captured for each node is also used to derive the workflow, in particular the input and output files that connects the different programs.

2.2 Unpacking Step

Given an experiment created in environment E, a reviewer needs to unpack and run it in a new environment E', similar to E.[1] The module Package Extraction (see Figure 1) is responsible for extracting the files and placing them in a single directory in E', i.e., no changes are made to other directories. The workflow is also automatically configured so that paths to programs and input files, and paths defined in environment variables, are adjusted to use the experiment directory in E'. Note that environment variables are configured *only* for the workflow execution – the original variables remain unchanged to avoid interfering with the normal operating environment of E'.

[1]Executables of the experiment will fail in E' if they are incompatible with the Linux kernel or hardware architecture.

2.3 Verification and Exploration

Users may run the experiment from the command line and examine the results. They can also run the workflow using VisTrails, and by doing so, they can leverage the mechanisms VisTrails provide to aid users in exploratory tasks, including the ability to perform parameter sweeps, to compare multiple results side by side, and to extend the original workflow. In addition, because VisTrails keeps detailed provenance of exploratory tasks, this provenance can serve as a means of communication between the reviewer and the authors.

Using `ReproZip`, (i) experiment execution is straightforward – reviewers need only load and "play" the workflow; (ii) there is a visual representation of the experiment, where main input and output files are explicitly described, which can help the reviewer better understand the structure of the experiment; (iii) reviewers can easily explore the experiment and try different parameters and input files; (iv) reviewers can also extend the original workflow to explore different techniques or perform analyses (e.g., generate plots) different from the ones produced by the authors; and (v) the provenance of the verification process is automatically captured by VisTrails – if the reviewer finds an issue with a given configuration, she can send the exact configuration back to the authors.

Figure 3 illustrates the verification and exploration process performed in marching cubes experiment. The original workflow is shown on the top left. The reviewer extended the workflow to derive a visualization, and after verifying the results, he also used the parameter sweep feature of VisTrails to compare results for multiple values for the isosurface. Examining the different isosurfaces enables the reviewer to verify the robustness of the marching cubes algorithm being evaluated.

3. DEMONSTRATION

In our demonstration, we will encourage demo visitors to use the tool themselves. We have used `ReproZip` to create reproducible packages for a number of (real) experiments in different domains, including molecular biology, visualization and database research. Visitors will choose one of the use cases we will provide and try all three parts of the reproducibility process: (i) experiment creation, (ii) packing on a source environment E, and (iii) unpacking and reproducing (with variations) on a target environment E'. For example, for the molecular biology use case, to create an experiment, visitors will generate random data characteristic of molecular biology experiments, decide among a set of network inference tools, decide how to put them together and generate a result. After running this experiment, they will be able to view the dependencies that have been created, both direct and transitive, and pack the experiment. Similarly, for the visualization experiment (topological correctness of marching cubes), visitors will be able to explore the marching cube algorithm and test its robustness by trying different values for the isosurface. In the third part of the reproducibility process, they will transfer the experiment to a different environment and run it. They will be able to vary some parameters for some of the tools to see how the results change.

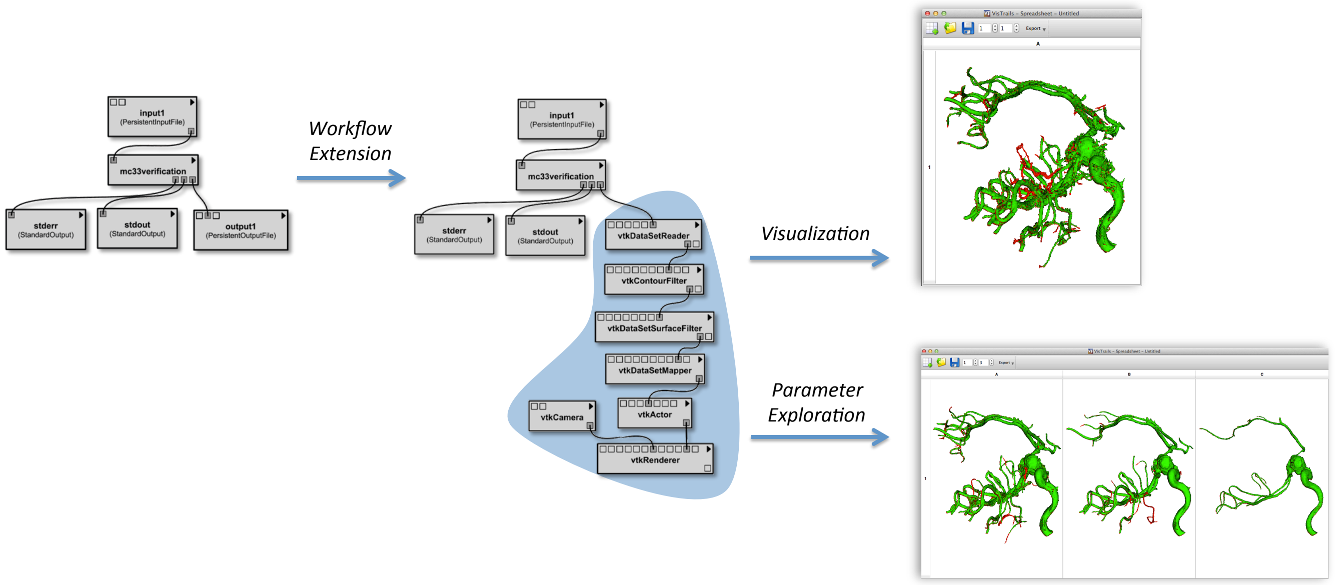

Figure 3: Verifying the topological correctness of a marching cubes algorithm. By using the workflow derived by ReproZip, the reviewer can extend it to visualize the results derived by the author. The reviewer can also verify the robustness of the algorithm by exploring different isosurfaces using the parameter sweep feature of VisTrails.

4. CONCLUSIONS

The perceived difficulty of packing experiments has discouraged authors from publishing reproducible results. Our system `ReproZip` simplifies this task. By combining features of scientific workflows and tools that transparently capture information about software and data dependencies, `ReproZip` not only simplifies the process required to create reproducible experiments, but it also helps reviewers to validate the results and communicate their findings to the authors. While our initial evaluation has shown that `ReproZip` is effective for a wide range of experiments in different domains, there are situations where the tool fails, e.g., when a given executable uses a hard-coded absolute path, or when the reviewer does not have an environment that is compatible with that of the author. For these cases, our current approach is to use `ReproZip` together with a virtual machine. We hope that as more authors adopt the practice of publishing reproducible results, they (as well as tool developers) will also adopt best practices that are conducive to reproducibility.

5. ACKNOWLEDGMENTS

We thank Jesse Lingeman for feedback on our initial ideas and for providing the molecular biology experiment we will use in our demonstration. We also thank Lis Custódio and Tiago Etiene for providing their marching cubes experiment that we have used to test `ReproZip` and will also offer to our demo visitors as a use case. This work was supported in part by the National Science Foundation (grants CNS-1229185, CNS-1153503, IIS-1139832, IIS-0905385, IIS-1142013, IOS-0922738, MCB-0929338 and MCB-1158273) and the National Institutes of Health (grants GM 32877-21/22 and 2R01GM032877-25A1).

6. REFERENCES

[1] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD*, pages 1345–1350, 2008.

[2] A. Davison. Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science Engineering*, 14(4):48 –56, july-aug. 2012.

[3] D. Donoho, A. Maleki, I. Rahman, M. Shahram, and V. Stodden. Reproducible research in computational harmonic analysis. *Computing in Science & Engineering*, 11(1):8–18, Jan.-Feb. 2009.

[4] J. Freire, D. Koop, E. Santos, C. Scheidegger, C. Silva, and H. T. Vo. *The Architecture of Open Source Applications*, chapter VisTrails. Lulu.com, 2011.

[5] J. Freire and C. T. Silva. Making Computations and Publications Reproducible with VisTrails. *Computing in Science and Engineering*, 14(4):18–25, 2012.

[6] GenePattern. `http://www.broadinstitute.org/cancer/software/genepattern/`.

[7] P. Guo. CDE: A Tool for Creating Portable Experimental Software Packages. *Computing in Science and Engineering*, 14(4):32–35, 2012.

[8] P. J. Guo and M. Seltzer. Burrito: wrapping your lab notebook in computational infrastructure. In *Proceedings of the 4th USENIX conference on Theory and Practice of Provenance*, TaPP'12, pages 7–7, Berkeley, CA, USA, 2012. USENIX Association.

[9] R. LeVeque. Python tools for reproducible research on hyperbolic problems. *Computing in Science & Engineering*, 11(1):19–27, Jan.-Feb. 2009.

[10] Madagascar. `http://www.ahay.org/wiki/Main_Page`.

[11] MongoDB. `http://www.mongodb.org/`.

[12] SystemTap. `http://sourceware.org/systemtap/`.