Supporting Thorough Artifact Evaluation with Occam

Luís Oliveira, David Wilkinson, Daniel Mossé, Bruce R Childers
Computer Science Department, University of Pittsburgh
{loliveira,dwilk,mosse,childers}@pitt.edu

Abstract

Efforts such as Artifact Evaluation (AE) have been growing, gradually making software evaluation an integral part of scientific publication. In this paper, we describe how Occam can help to mitigate some of the challenges faced by both authors and reviewers. For authors, Occam provides the means to package their artifacts with enough detail to be used within experiments that can be easily repeated. For the reviewers, Occam provides the means to thoroughly evaluate artifacts by: allowing them to repeat the author's experiments; providing the ability to modify inputs, parameters, and software to run different experiments.

1 Introduction

Academic and scientific productivity is often measured by peer-reviewed publications. These papers typically rely on software artifacts to produce a significant part of the research cycle for any results published. However, software is usually neither examined nor credited with the same standards as the papers.

Despite that, efforts such as Artifact Evaluation (AE) have been growing, gradually making software evaluation an integral part of scientific publication. Some benefits of AE include: (1) authors can get credit for the time they spent developing software, (2) reviewers can test the paper claims, reproducing results or even testing beyond what the paper presents, and (3) readers can have a higher degree of confidence in the paper they are reading. Other gains are less obvious, but not less important: (4) authors follow a process that improves software quality (including documentation and instructions on how to use), (5) AE is a good motivator to package software artifacts for distribution. Distributing artifacts (6) improves communities by giving access to software that helps speed up research, and (7) to authors by increasing the use of their software (and citations).

Running the same experiment as presented and obtaining the same results is only the first step for Artifact Evaluation (AE). Thorough reviewers may want to run different experiments by trying different inputs that were not considered in the original experiment (e.g., their own benchmarks, different parameters, or community-developed benchmarks); or may even want to build the artifact with a different compiler or library version and ensure results hold.

Within AE, the problem is twofold: it is hard to evaluate artifacts, and it is even harder to create good artifacts. To provide reviewers the ability of doing such a thorough evaluation, an AE system *must do better than a tar file with binaries and datasets.* Instead, such a system should provide authors with the ability to easily create artifacts that can be executed by others effortlessly.

In this paper, we describe how Open Curation for Computational Artifact Management (Occam)'s design of experimental workflows and software packaging system helps solving these problems for both authors and reviewers. First, we explore how Occam's experimental workflows can improve they way authors distribute their experiments, and improve reviewers evaluations by providing a simple interface to a complex problem. Then, we discuss why packaging software using Occam can improve the quality of artifacts. In particular how it helps artifacts to be easily deployed, inspected, and modified by the reviewers

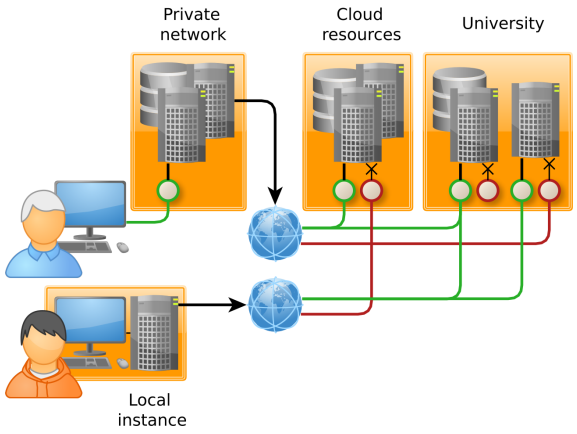

Figure 1: Occam's federation: Multiple resources linked

to produce thorough evaluations.

2 Occam overview

Occam is a community-driven active curation platform that allows its users to contribute and share their artifacts and experiments. Occam features a multi-instance organization that marshals contributed computational and storage resources into a federation (see Figure 1). Any user can instantiate their own local instance of Occam (e.g., in their workstations) that can use private local resources to store artifacts, and to execute experiments. Occam's federated design also allows connecting the user's local instance with other private or public instances to access a pool of community shared tools (such as simulators or benchmarks).

Occam is designed to satisfy the requirements for long-term reproducibility of software experiments [1], which overlaps with AE. In particular, Occam soundly preserves software artifacts by including the source code alongside build and run instructions. Furthermore, Occam preserves all artifact dependencies (recursively) with the same level of detail. Occam artifacts can be composed together in the form of experimental workflows. Through these workflows, Occam keeps a complete record of the experimental dataflow to allow the re-execution of the experiment. Results from experiments can also be shared like all content curated in Occam. Artifacts and experiments can be modified and reused including changes to input parameters and source code.

The system is currently implemented and deployed in a demo server (at https://occam.cs.pitt.edu).

3 Experimental workflows

Occam was designed for *executable experimental workflows*. In essence, an experimental workflow is a directed acyclic graph that describes the processing of input data/configurations by software. Figure 2 depicts a simple workflow as seen in Occam's workflow editor: a memory simulator "DRAMSim2" [2] with an input that represents a trace that the simulator can run (connected to the simulator in the workflow). The disconnected port to the right of the simulator represents the output of the workflow.

Occam orchestrates the execution of workflows by issuing jobs for each component contained within. When each job finishes, Occam compiles the outputs and preserves them in its repository with provenance and lineage information. This information can then be used to inspect the software and datasets involved in the creation of the results, an invaluable trait for AE. Once intermediate outputs are added to Occam's repository, any jobs that were waiting for those outputs for further processing are triggered automatically; until the experiment completes.

Users (depending on permissions) can inspect, modify and re-execute an experimental workflow. For example, in Figure 2, a reviewer could remove the trace file (on the left) and replace it with a different

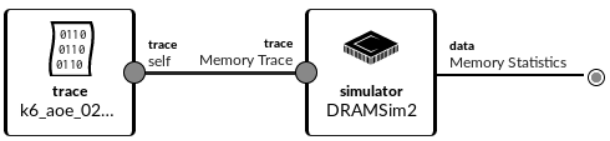

Figure 2: Workflow creation tool, a trace file (left) is connected to a simulator (right).

DRAMSim2 trace; then, the reviewer would execute the workflow with the click of a button and obtain new results.

4 Packaging artifacts

An artifact that is contained within Occam is called an *Occam object*. As part of the process of preparing such an artifact, the developer of that object needs to describe the software or data that will packaged.

Each object has an intrinsic identifier that uniquely identifies it across any instance of Occam. Occam also tracks each change made to an artifact through the use of 'git', a commonplace version tracking tool, through the behind-the-scenes management of a git repository for each object. This allows interaction with older versions of the artifact and a view of its history. For instance, one could look at the exact version of an artifact used for a paper even if other changes have since been made, which is a particularly useful feature for scientific production. In fact, these two aspects of artifact identification (unique ids and version) are automatically recorded directly in the experimental workflow.

Also visible in the experimental workflow, are the inputs and outputs of each artifact. These are also described by the developer as part of the process of packaging the artifact and allow automatic checking of workflow semantics.

A proper artifact for AE must be more than a tar file with the software and datasets and should include (a) source code for reviewers to inspect, (b) instructions to build the artifact to allow reviewers to modify the source code and rebuild it, (c) instructions to install the artifact to ensure the replicability of the execution environment, (d) instructions to run the artifact to guarantee the correct commands are issued, and (e) a thorough list of all software the artifact depends on (i.e., dependencies).

There are many tools that provide the means to package artifacts for AE, however, they do not fulfill all of these requirements. ReproZip [3] and CDE [4] focus on repeating the exact same experiment multiple times, thus they only creates a static package containing files that were used during the original experiment, which hinders the ability to run different experiments (e.g., with different inputs). As such, there is no list of other software dependencies. Umbrella [5] explicitly rejects the necessity of preserving the "mess" (i.e., the code) and instead only preserves binaries. They do realize the importance of dependencies, as such they document and package those, however, the lack of source code means that reviewers cannot inspect and modify the software for a thorough review, and some important information is not preserved (e.g., how software was compiled). A similar approach is taken by Pachyderm [6] and Whole Tale [7], preserving a virtual machine (VM) and using it to process input data and generate results. This approach requires the preservation of a VM for each artifact, which can quickly become cumbersome, and these machines are not designed to preserve and rebuild the source code of artifacts.

Gathering all the information about software dependencies is not an easy task for the non-expert. For example, a researcher creates a Python script to process the collected data and may know Python 3 is required, because the script was programmed explicitly using that language. But the researcher may not know that Python requires an XML parsing library to process the data or the exact version of Python used in the experiment. Occam's approach to managing dependencies greatly improves reproducibility and portability in these cases. To remove the burden of having the developer specify the complete hierarchy of dependencies, Occam recursively discovers all the dependencies an artifact requires to execute. Then, developers only need to specify the dependencies of the software they are packaging; those known and specific dependencies will have been packaged by developers. This implies that Occam objects can only depend on other Occam objects, and if an artifact requires software that does not exist in the Occam repository, the developer needs to first package that software explicitly into an Occam object. Nevertheless, this process enhances the software development an makes it more reproducible, usable, and portable.

Occam handles the execution of artifacts by creating a digest of all software the artifact requires (recursively). Then, it creates a VM based on that digest artifacts [8], and executes (either build or run) the ar-

tifact. This process relies on developers providing the build and run scripts as part of the process of packaging their software. However, the end-user (e.g., a reviewer) does not need to be aware of them unless s/he deems necessary to inspect them (e.g., to inspect compilation flags). These scripts are very similar to what developers already create for their software, and the only modification required by Occam is the location of inputs and outputs. To create the VM, Occam follows developer instructions on how to install the artifact, namely what environment variables need to be set in the running environment and where files need to be placed (e.g., binaries in `/usr/bin` and libraries in `/usr/lib`). The benefit of this process is that execution will fail early if the artifact depedencies are not complete or if any of the build/run/install instructions are incorrect or incomplete.

5 Conclusion

As Artifact Evaluation keeps growing within the scientific community, the pressure to come up with reliable solutions to the problems faced by reviewers and authors increases. Due to its background in long-term reproducibility, Occam provides a system that authors can leverage to create portable artifacts and experiments that can be easily deployed, used, and modified by reviewers/users that want to thoroughly evaluate or experiment with artifacts. In particular, Occam provides a packaging mechanism that allows authors to include their software, bundled with all the instructions on how to build, run, and install it; and to maintain a complete list of software dependencies. Moreover, by preserving the provenance and lineage information together with any generated outputs, it allows the inspection of software and datasets involved in the creation of experimental results.

References

[1] L. Oliveira, D. Wilkinson, D. Mossé, and B. Childers, "Supporting long-term reproducible software execution," in *Proceedings of the First International Workshop on Practical Reproducible Evaluation of Computer Systems*, ser. P-RECS'18. New York, NY, USA: ACM, 2018, pp. 6:1–6:6. [Online]. Available: http://doi.acm.org/10.1145/3214239.3214245

[2] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[3] "ReproZip - About," https://vida-nyu.github.io/reprozip/, [Online; accessed 29-Aug-2016].

[4] "CDE: Lightweight application virtualization for Linux," http://www.pgbovine.net/cde.html, [Online; accessed 29-Sep-2017].

[5] H. Meng, D. Thain, A. Vyushkov, M. Wolf, and A. Woodard, "Conducting reproducible research with umbrella: Tracking, creating, and preserving execution environments," in *2016 IEEE 12th International Conference on e-Science (e-Science)*, Oct 2016, pp. 91–100.

[6] "Reproducible Data Science that Scales! - Pachyderm," https://pachyderm.io, 2018, [Online; accessed 9-Apr-2018].

[7] A. Brinckman, K. Chard, N. Gaffney, M. Hategan, M. B. Jones, K. Kowalik, S. Kulasekaran, B. Ludäscher, B. D. Mecum, J. Nabrzyski, V. Stodden, I. J. Taylor, M. J. Turk, and K. Turner, "Computing environments for reproducibility: Capturing the "whole tale"," *Future Generation Computer Systems*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17310695

[8] D. Wilkinson, L. Oliveira, D. Mossé, and B. Childers, "Software provenance: Track the reality not the virtual machine," in *Proceedings of the First International Workshop on Practical Reproducible Evaluation of Computer Systems*, ser. P-RECS'18. New York, NY, USA: ACM, 2018, pp. 5:1–5:6. [Online]. Available: http://doi.acm.org/10.1145/3214239.3214244