Reproducibility responsibilities in the HPC arena

Mark R. Fahey
University of Tennessee, Knoxville
Joint Institute for Computational Sciences
mfahey@utk.edu

Robert McLay
University of Texas at Austin
Texas Advanced Computing Center
mclay@tacc.utexas.edu

April 1, 2014

Abstract

Expecting bit-for-bit reproducibility in the HPC arena is not feasible because of the ever changing hardware and software. No user's application is an island; it lives in an "HPC" eco-system that changes over time. Old hardware stops working and even old software won't run on new hardware. Further, software libraries change over time either by changing the internals or even interfaces. So bit-for-bit reproducibility should not be expected. Rather a reasonable expectation is that results are reproducible within error bounds; or that the answers are "close" (which is its own debate.)

To expect a researcher to reproduce their own results or the results of others within some error bounds, there must be enough information to recreate all the details of the experiment. This requires complete documentation of all phases of the researcher's workflow; from code to versioning to programming and runtime environments to publishing of data. This argument is the core statement of the Yale 2009 Declaration on Reproducible Research [1]. Although the HPC ecosystem is often outside the researchers control, the application code could be built almost identically and there is a chance for "very similar" results with just only round-off error differences. To achieve complete documentation at every step, the researcher, the computing center, and the funding agencies all have a role. In this thesis, the role of the researcher is expanded upon as compared to the Yale report and the role of the computing centers is described.

1 Overview

One definition of *reproducibility* is the ability of an entire experiment or study to be reproduced, either by the researcher or by someone else working independently. With the ever changing landscape of hardware and software, and in HPC the ubiquitous of custom hardware and software, all of which that has relatively short-lived timespans, reproducibility beyond a few years is a difficult proposition if not infeasible.

The Yale report [1] referred to this topic as "reproducible computational research" and said that "scientists make all details of the published computations (code and data) conveniently available to others". Much of what they suggest is highly commendable and more importantly very doable. Complete documentation from code to versioning to programming and runtime environments to publishing of data would allow other researchers to reproduce everything about the researcher's code and how it was built even given the barriers above. The hardware and other parts of the software environment are outside the researchers control, but the application code could be built almost identically and there is a chance for "very similar" results with just only round-off error differences.

To achieve complete documentation at every step, the researcher, the computing center, and the funding agencies all have a role. In the following sections, the researcher's responsibilities are discussed (relative to the Yale 2009 Declaration) and the role of the computing centers are discussed which is completely absent in the Yale report. From this point on, reproducibility means recreating the published experiment using all the available documentation.

2 What researchers should do

In the Yale 2009 Declaration [1], there are many worthwhile recommendations for scientists such as: (1) provide links to source-code and input data, (2) have a unique ID for each version of released code and input data [implied but not explicitly said that the researcher should only generate data with released code version], (3) include information describing the computing environment and potentially include a virtual machine, (4) use open source licensing to facilitate reuse, (5) use an open access contract for published papers, and (6) publish data and code in nonproprietary formats. These recommendations as a whole are commendable and yet there are nontrivial subtle issues that make some of the recommendations not obvious to solve.

First and foremost, one recommendation the report does not explicitly state, though it is implied, is that the the researcher should document everything with what is called good coding practices. As in any field, documentation is the key. Each function or object should have a header describing what it does or is and its inputs and outputs or methods, and there should be a scattering of comments throughout the code explaining key parts. This should be done for every part of the code and there should also be a paper that describe how the code is designed, the underlying mathematics, physics, and computer science, and there should be a manual describing how to use the code and the input and output. This is by far the most important piece of scientific reproducibility because it is highly coupled with the scientific integrity of the code.

The second recommendation from the Yale report that is highly commendable – unique ID for each version of a code. This recommendation should include the following steps which are all important:

1. Version software. There are many versioning options (like git, svn, cvs, etc) available to researchers. And even if the researcher does not want to use a framework, they can still do their own manual versioning with releases. And most importantly the research must only use released versions of the software to generate results, not developmental versions.
2. Document each version of a library or an application used by their software including MPI
3. Document the version of the compiler used.
4. Document the version of the OS. This is subtle because although you can identify the OS release, knowing each and every patch applied is harder to document. Even more complicating is the fact that some machines have two or more operating systems that they support (for instance one for login nodes and one for compute nodes.) This is where expecting a researcher to know all the things to document may be too much to assume.
5. Document the hardware resource (CPUs, interconnect, memory size and type). Again, this may be asking too much of a researcher, especially on complicated, hybrid machines.
6. Document exactly how the code was run; batch job options, job launch options, every environment variable setting (some have direct affect on how MPI works). This a researcher should be able to do.

A researcher can do some of the above without too much additional overhead and data requirements; and this information should become a part of every experiment run and paper published. As noted above, the researcher might need some help from a center to know the proper compile- and run-time data to document. If a researcher were to do all the above, reproducibility of results becomes quite attainable.

Note that as long as the OS for that machine hasn't changed AND the versions of the compiler and software are still available AND the machine hardware hasn't changed, bit-for-bit reproducibility is very likely. As soon as the OS changes or the hardware is upgraded, then one can only expect to get close answers. What if the answers are clearly wrong? The centers managing HPC resources aren't going to revert an OS or upgrades just because you got different results unless the changes are proven to be defective in some way. In fact, OS upgrades happen quite often especially if there are security concerns (which typically trumps everything else.) And hardware is upgraded as often as budgets allow. So there are still potential issues.

2.1 A VM of the computing environment - issues

Recommendation 3 from the Yale report suggests that the researcher potentially include a virtual machine (VM) that contains the computing environment where the results were generated. Given the above discussion, I hope it is obvious that within HPC environments that this recommendation is infeasible. A researcher cannot be expected to create a virtual machine replicating the environment on a national computing resource that takes several administrators to run. Further, this recommendation is infeasible from the standpoint of data requirements – creating and storing a VM for each and every run would require a large amount of disk space. And yet another problem is that even if you could make the VMs and store them for extended periods of time, will they even work on hardware several years down the road when the machine they ran on no longer exists? For vendor specific software (OS or even third party libraries with modifications) provided by vendors (Cray, SGI, IBM, etc), just 5 years down the road each and every version of the software that researchers had access to may not exist or at least could be very hard to track down. See the recent article on running the first Cray OS on [2]. This does not even solve the problem with vendor specific software which they may not share.

A researcher could attempt to do all the above by storing relevant software (OS, libraries, compilers), But this seems too much to ask individual researchers to store especially if they are just a "user" of some center's cluster (managed by others) and even a bit much if they do manage it themselves. The next section provides an alternate solution and puts the onus on the computing centers.

3 The role that computer centers can and should provide

There are many worthwhile recommendations for scientists and for funding agencies and for journal editors in the Yale 2009 Declaration [1], however there are no recommendations for computing centers. This is a glaring hole as I believe the centers (along with the funding agencies) have a obligation to solve some parts of the reproducibility issue of gathering all relevant documentation.

Computing centers (at the university and national level both) can address if not solve repository and software versioning issues for researchers by providing snapshots of the OS (documentation of the versions) and libraries and providing views into databases holding these snapshots. Since centers control the system software versions and the available third party software stack, it would make the most sense if the centers would at the very least document each and every version of all of their software and the duration it was the default on the machine. There are already ongoing efforts to capture most of this information at some centers. For example, at the National Institute for Computational Sciences, much of the programming environment software versioning is documented and the center can provide users a list of all the system defaults at any time from the past and even have “all-in-one” modules that provide these snapshots.

Similarly, it would be of great benefit to the community if centers would make RPM bundles of the system software and hopefully provide a test bed cluster with which one could “revert” to past system software installations to confirm reproducibility (at least for the life of the technology/award.) However, test bed clusters are sometimes not part of HPC deployments although they should be. And the test bed cluster would likely be only a few nodes, so reproducing large simulations would not be feasible.

However, this does not solve the problem of documenting the software used by the researcher who might use non-default packages. What is needed is an automatic way to collect the information on the versions of software used by the researcher. This is what the centers (national or at the campus level) should be providing. For example, NICS and TACC provide two similar but slightly different prototypes (ALTD [3] and Lariat [4], respectively) that capture the libraries and their versions used by each researcher for each code they build and run. This exactly solves part of the documentation problem; in fact NERSC uses the NICS infrastructure (ALTD) and added functionality for users to find out this provenance data from old builds so they can build their codes exactly like they did months or years before. Note that a new effort (called XALT - under development) is underway to combine and extend these infrastructures from NICS and TACC to capture even more information - basically almost everything mentioned above.

Every center should be doing this for a variety of reasons from better user support to provenance data collection to security related concerns. Collecting this information is very doable (as proven by the prototypes) and has been proven to be very useful. It would help the researchers greatly with providing the information the Yale report recommends.

4 Recommendations

Researchers can and must do version control (either with tools or even tar balls with version number) and they must document the computing environment in which they ran (to the extent possible including third party libraries, compilers, operating system, hardware used, and run time options.)

Computing centers must provide a data into all the libraries that were available (and the defaults) as well as automatically tracking all the libraries and runtime options used – many do not do this, but it is already possible with two prototypes (ALTD and Lariat) and a new combined effort underway (XALT.)

References

[1] Yale Law School Roundtable on Data and Code Sharing, *Reproducible Research: Addressing the need for data and code sharing in computational science*, Computing in Science & Engineering, IEEE CS, 2010.

[2] http://www.theatlantic.com/technology/archive/2014/01/these-two-guys-tried-to-rebuild-a-cray-supercomputer/283071/

[3] Mark Fahey, Nick Jones, and Bilel Hadri, The Automatic Library Tracking Database, In *Proceedings of the 52nd Cray User Group (CUG10)*, May 2010.

[4] Robert McLay Lariat Repository, 2013. https://github.com/TACC/lariat.