A Portable Approach for Repeatability Testing

[Extended Abstract]

Quan Pham
quanpt@cs.uchicago.edu

Tanu Malik
tanum@uchicago.edu

Ian Foster
foster@anl.gov

ABSTRACT

Provenance-To-Use (PTU) is a tool from SOLE Framework that minimizes computation time during repeatability testing of computation-based experiments. PTU allows authors to assemble code, data, environment, and provenance of an initial reference execution into a single package that can be distributed to testers. The resulting package allows testers to view the provenance graph and specify, at a process and file level, nodes of the graph that they want for a partial deterministic replay-based, for example, on their compute, memory and I/O utilization as measured during the reference execution. Using the provenance trace, PTU guarantees that events are processed in the same order using the same data from one execution to the next. We show the efficiency of PTU for conducting repeatability testing of workflow-based scientific programs outweighs the overhead incurred during the reference executions.

1. INTRODUCTION

Increasingly conference committees and journal editors are encouraging authors to submit their code, data and software for repeatability testing. Repeatability testing improves peer review in three ways: (1) It allows reviewers to not only read the ideas in the paper, but validate them by running the accompanying software; (2) It allows reviewers to work the software for different kinds of data and parameters to check robustness of the idea; and finally (3) It allows reviewers to determine the limitations and assumptions in the ideas by checking for boundary conditions.

However, as documented by recent experiments, repeatability testing can be arduous and time consuming for both authors and testers [5, 8]. Authors have to prepare code, document it, and make explicit rendering of all dependencies on compilers, operating systems and hardware. For testers, assessing code and data for repeatability can be challenging since documentation is rarely complete and perfect. But more so, as experiments become data and computation intensive, the testing time can be significant [12].

2. PTU: PROVENANCE-TO-USE

Provenance-To-Use (PTU) [10] is a portable SOLE [11] tool for reducing time for repeatability testing. Authors can use PTU to accomplish two tasks: (1) build a package of their source code, data, and environment variables, and (2) store process and file level details about a reference execution of their system in an accompanying database. A package alleviates testers from software deployment issues allowing for convenient distribution. Preserving a reference execution path and run-time details within a package eases distribution of this vital information that can guide testers during the testing phase. In particular, testers can explore the provenance graph and accompanying run-time details to specify the part of the provenance graph that they want to re-execute or replay.

In order to acquire the information to build its package, PTU uses a tracing mechanism on authors' experiment. There are different tracing mechanisms at user-space level, such as ptrace [7] or kernel-based mechanisms such as SystemTap [3]. Kernel-based mechanisms have better performance compared to the user-space tracing mechanisms due to the fact that they avoid context switching between tracees and tracer [7]. However from a reproducibility standpoint, they also have significant drawbacks. SystemTap needs to run at a higher privilege level; it requires root access, creating a more restricted environment. It also requires a Linux kernel debuginfo package that must correspond to the *exact kernel version* installed on the host system. These requirements, we believe, strongly affects the portability of an application since often users do not have root access. To provide portability to PTU package, we use ptrace tracing mechanism.

Relying on Unix *ptrace* system call interposition, CDE [6] can collect code, data files, and environment variables to create and run a package. The *ptrace* mechanism also allows for auditing file and process information, which can be transformed for storing a provenance graph, independent of the application [9]. We enhance the *ptrace* mechanism in CDE to store a provenance graph representing a reference run-time execution at the process and file level. PTU allows authors to use *ptu-audit* to create a self-contained package with a reference execution by prepending the application command with the *ptu-audit* tool.

The *ptu-audit* tool internally uses *ptrace* to monitor about 50 system calls including process system calls, such as *execve*(), *sys_fork*(); file system calls, such as *read*(), *write*(), *sys_io*() for collecting file provenance; and network calls, such as *bind*(), *connect*(), *socket*(), and *send*() for network auditing. Whenever system calls occur, PTU notes the identifier of the process that made the system call to extract more information about the process from the Linux */proc* file system. A separate thread is used to obtain memory footprint, CPU consumption, and I/Os of the process from */proc/$pid/stat* every three seconds.

When a system call (file or network) returns, PTU emulates CDE functionality. In the case of a file, it copies the accessed file into a package directory that consists of all sub-directories and symbolic links to the original file's location. In the case of a network it copies the network connection information including IP and port information. The package directory also contains the SQLite database that stores the provenance information of the test run. When the entire test run finishes, PTU builds a reference execution file consisting of the topological sort of the provenance graph. The nodes of the graph enumerate run-time details such as process memory consumption, and file sizes. The tester, as described next, can utilize the database and the graph for efficient re-execution.

Testers can repeat the execution in a selective manners using

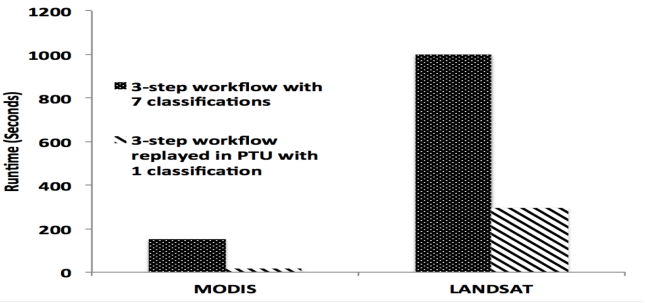

Figure 1: Time reduction when PTU is used to repeat workflows for $PEEL_0$

ptu-exec tool. Prior to running a package a tester can view the provenance graph of the reference execution. This graph can be viewed at the granularity of processes and files, and can aid the tester in visually determining parts of the program that they wish to re-execute. To specify a re-execution, a tester can either specify nodes on the provenance graph or modify a run configuration file that is included in the package.

A key distinguishing characteristic of our approach is that it can provide fast re-execution and correct output even when no workflow definition or management system is in place. PTU does not force testers to keep mental track of processes that were run, while allowing them to use simple command-line options. Our design does assume that provenance meta-data is complete and does not omit, alter, or create false provenance events, and is stored persistently.

3. USE CASE AND EXPERIMENTS

To test PTU we used a paper from Best et. al. [1] in which authors shared data, tools and software. We constructed meaningful testing scenarios and determined if PTU provides any performance improvements. The paper describes the synthesis methodology to generate a new land cover dataset for the conterminous USA called $PEEL_0$. The methodology is implemented as an R-program, implemented as a three-step workflow process, in which the second step is to execute a model which inputs among other parameters a classification scheme. This step is memory-intensive and a typical run takes close to a GB of real memory. To reduce memory consumption, testers may want to further lower the spatial resolution by specifying even fewer input files and a simpler classification model.

Figure 1 show the performance improvements of using PTU with $PEEL_0$. The most important statistic is the slow down in running the process during a test execution at about 35% for $PEEL_0$. This slowdown is due to additional system call tracing, with many file system calls traced for $PEEL_0$. These slowdown is easily offset during the re-execution phase. $PEEL_0$ has a significant improvement (>71%) since the entire process is run on smaller input files.

4. RELATED WORK

There are different approaches to the problem of repeatability testing. CDE [6] helps authors package the code, data, and environment for running and deploying on other machines without installation or configuration. ReproZip [2] uses kernel-based tracing SystemTap to captures detailed provenance of existing experiments and combine into a package that can be installed and run on a different environment. To overcome incompatibility of the Linux kernel, executable using a hard-coded absolute paths, ReproZip is used together with virtual machines.

Kameleon [4] tool is a virtual machine based approach. It is a bash script generator designed to create initial virtual images from scratch for any linux distributions. Using recipes, users can generate customized virtual images with predefined software packages to run on different cloud computing service providers. This tool is complementary to our work and we plan to integrate PTU with it.

5. CONCLUSIONS

PTU is a step toward testing software programs that are submitted to conference proceedings and journals to conduct repeatability tests. Peer reviewers often must review these programs in a short period of time. By providing one utility for packaging the software programs and its reference execution without modifying the application, we have made it easy and attractive for authors to use it and a fine control, efficient way for testers to use PTU. Currently, the implementation of PTU is being extended to support distributed computational experiments. PTU will capture data flows and network state among participating compute nodes to allow selective replay of distributed experiments as well as replay of distributed experiments on limited resources.

6. REFERENCES

[1] N. Best, J. Elliott, and I. Foster. Synthesis of a complete land Use/Land cover dataset for the conterminous united states. *SSRN eLibrary*, 2012.

[2] F. Chirigati, D. Shasha, and J. Freire. Reprozip: using provenance to support computational reproducibility. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*, TaPP '13, pages 1:1–1:4, Berkeley, CA, USA, 2013. USENIX Association.

[3] F. C. Eigler and R. Hat. Problem solving with systemtap. In *Proc. of the Ottawa Linux Symposium*, pages 261–268, 2006.

[4] J. Emeras, O. Richard, and B. Bzeznik. Reconstructing the Software Environment of an Experiment with Kameleon. Rapport de recherche RR-7755, INRIA, Oct. 2011.

[5] J. Freire, P. Bonnet, and D. Shasha. Computational reproducibility: State-of-the-art, challenges, and database research opportunities. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 593–596, New York, NY, USA, 2012. ACM.

[6] P. J. Guo and D. Engler. CDE: using system call interposition to automatically create portable software packages. pages 21–21, Portland, OR, 2011. USENIX Association.

[7] J. Keniston, A. Mavinakayanahalli, P. Panchamukhi, and V. Prasad. Ptrace, utrace, uprobes: Lightweight, dynamic tracing of user apps. In *Proceedings of the 2007 Linux Symposium*, pages 215–224, 2007.

[8] S. Manegold, I. Manolescu, L. Afanasiev, J. Feng, G. Gou, M. Hadjieleftheriou, S. Harizopoulos, P. Kalnis, K. Karanasos, D. Laurent, and others. Repeatability & workability evaluation of SIGMOD 2009. *ACM SIGMOD Record*, 38:40–43, 2010. 3.

[9] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. pages 4–4, Boston, MA, 2006. USENIX Association.

[10] Q. Pham, T. Malik, and I. Foster. Using provenance for repeatability. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*, page 2, 2013.

[11] Q. Pham, T. Malik, I. Foster, R. D. Lauro, and R. Montella. SOLE: linking research papers with science objects. Proceedings of the 4th International Workshop on Provenance and Annotation, 2012.

[12] D. Sasha. Repeatability & workability for the software community: Challenges, experiences, and the future.