Using Virtualisation For Reproducible Research And Code Portability

Svetlana Sveshnikova Ivan Gankevich
Dept. of Computer Modeling and Multiprocessor Systems
Saint Petersburg State University
Saint-Petersburg, Russia
Email: st012967@student.spbu.ru, i.gankevich@spbu.ru

EXTENDED ABSTRACT

***Abstract*—Research reproducibility is an emerging topic in computer science. One of the problems in research reproducibility is the absence of tools to reproduce specified operating system with specific version of the software installed. In the proposal reported here we investigate how a tool based on lightweight virtualisation technologies reproduces them. The experiments show that creating reproducible environment adds significant overhead only on the first run of the application, and propose a number of ways to improve the tool.**

***Index Terms*—Linux namespaces, Linux cgroups, compiler tools, lightweight virtualisation.**

I. Introduction

Research reproducibility is an emerging topic in computer science [1], [2]; although, repeating a research work in computer science is often easier than in other sciences — one needs only a decent computer and the source code to reproduce the research — it may take a considerable amount of time to fully configure the platform: setup virtual or physical cluster, install compatible versions of operating system, software libraries and tools and compile and run the source code of the research work. Not only the source code, accompanying the paper, is published rarely, but it requires certain platform configuration to compile and run.

There are several stages on each of which (ideally) there should be a tool that automates reproducibility:

- hardware stage (finding the required hardware),
- operating system stage (installing compatible operating system),
- software stage (compiling and executing the programme),
- graphical stage (gathering statistics from programme runs and plotting graphs),
- publication stage (writing and publishing the paper with all the data, graphs and the source code included).

In this proposal we deal with operating system and software stages — automate creation of environment to compile and run the programme in. For this purpose we use lightweight virtualisation technologies (Linux namespaces) on the example of distributed batch processing programme that runs on a cluster of nodes and processes the data in parallel. Our tool, called *Collector*, creates root file system with the specified version of Linux distribution, the compiler and all the dependent packages. Then it compiles and runs the source code inside this virtual environment. The resulting root file system is portable across any platform with the same processor architecture and compatible kernel version.

The advantages of using raw file system over opaque operating system images are clear:

- It is portable: can be stored as is or in the archive, and converted to/from any OS image format.
- It can be mounted over cluster network to any number of cluster nodes, and used concurrently by several parallel processes via Union/Overlay file system.
- It can be directly patched/upgraded by changing the current root directory to the path of the raw file system.

The objective of the study reported here is to develop a tool that automates creation of such portable environments and makes building particular source code inside it repeatable regardless of underlying operating system. This is the first publication on this tool.

II. Related work

The topic of research reproducibility is active not only in computer science, but also in statistics. For example, in [3] the authors propose to use Org-mode — a plain text markup language — to insert source code of research immediately in the text of the paper and execute the code on every document export to produce tables and graphs. Although, the system is capable of running arbitrary scripts, it is impractical to include any real C/C++/Fortran source code, as it is generally large compared to the code that produces graphs and requires certain libraries/compilers to build and run. So, Org-mode support for reproducing graphs and tables is limited to relatively small programmes written in high-level languages (R/python/graphviz), that takes input data, produced elsewhere, and generates a graph or a table.

In [4] the authors discuss the importance of research reproducibility in parallel computing to improve trustworthiness of the experiments. The issues that prevent wide spread of reproducible research practice include

- the impossibility to reproduce research if someone uses unique hardware,
- publishing rules and agreements,
- the impossibility to obtain the required version of software.

So, the external rules and regulations may prevent publishing the whole paper together with the source code, but may not prevent publishing gathered data and the source code that produces the numbers.

Another idea is that its is not the source code that should be included in scientific paper, but that data, programme code and presentation of research may be stored together in a single file. This approach was explored in [5] where the authors suggest using Java Virtual Machine (JVM) to execute bytecode and HDF5 file format to store all the experimental data, source code and scripts for generating tables, plots and figures. Potential problems consist of using another programming languages, that are not supported by JVM (C/C++/Fortran), and storing large datasets in HDF5 file.

III. Collector tool

All computer science research works can be divided into two broad categories. On one side there are experiments with software or algorithms. In this research the most valuable part is the source code and configuration of execution environment, which usually consists of some operating system, processor architecture and software packages. On another side there are experiments with configuration of compute nodes and cluster network. To store and later reproduce operating system and execution environment we propose to use Collector — a programme that builds C/C++/Fortran source code by downloading and installing system packages in a separate root file system directory without super user privileges.

The task is accomplished via instantiating new mount and user Linux namespaces in which the original user is mapped to the super user. After that a new process is launched having all super user privileges inside these namespaces, and installs packages specified in the configuration file into the specified root file system directory. Finally, the current file system root is moved to this directory, a directory with the source code is mapped from the original root file system to the new one, and the code is compiled and run inside it.

Root file system that was created during the first run is saved, and subsequent runs of the application and code compilation do not cause installation of system packages (unless specified in the configuration file). On the first run Collector downloads and installs system packages specified in the configuration file from OS repository via package manager. It is a simple prototype that may be improved in future. For example, using network Linux namespace it is straightforward to run the application over virtual network with specified number of nodes and IP address range. Another improvement is to use Linux control groups to limit resource usage of each parallel process to make performance of virtual network more predictable. In our example we use CentOS operating system with RPM package manager, but the procedure can be adapted for other platforms.

In the experiment we compile and run the test programme [6] two times. During the first run Collector downloads and installs all the dependencies before compiling and running, and during the second run it only checks that dependencies are satisfied. After that it compiles the programme and runs tests. The experiment showed that initialising a separate root file system takes considerable amount of time compared to the execution time of tests, whereas subsequent runs are faster as they use already initialised environment (Table I). Performance-wise it would be more efficient to store read-only base image of the operating system in cache directory and use Union/Overlay file system to mount it under writable directory to reduce initialisation time.

TABLE I
Performance of root file system initialisation.

Action	Time, s	
	Exp. I	Exp. II
Download and install dependencies	548	9
Execute example	723	723
All time	1271	732

There are a number of potential problems that are related to lightweight virtualisation technologies. First, it requires recent version of Linux kernel (at least 3.10 fully supports all namespaces) to use unprivileged user namespaces, and this version is newer than the one that is widely used in HPC clusters. For example, Scientific Linux distribution, which is popular in GRID computing, uses kernel version 2.6.32, which is not capable of creating user namespaces. Second, lightweight virtualisation is available on Linux only, there is no compatible version of the technology neither for UNIX nor for POSIX-compliant operating systems.

IV. Conclusion

One of the problems in research reproducibility is the absence of tools to reproduce specified operating system with specific version of the software installed. Lightweight virtualisation technologies is a solution to this problem, that uses unprivileged Linux namespaces to create such execution environment in a separate root file system directory and package it together with the source code of the programme and its binary form. The solution does not pollute host operating system with programme dependencies and does not require super user privileges to create the environment. The future work is to investigate how network Linux namespace and control groups can improve application execution inside the environment.

References

[1] R. J. LeVeque, "Wave propagation software, computational science, and reproducible research," in *Proc. Int. Congr. of Mathematicians*, 2006.

[2] A. P. Davison, M. Mattioni, D. Samarkanov, and B. Teleńczuk, "Sumatra: a toolkit for reproducible research," *Implementing reproducible research*, vol. 57, 2014.

[3] E. Schulte, D. Davison, T. Dye, C. Dominik *et al.*, "A multi-language computing environment for literate programming and reproducible research," *Journal of Statistical Software*, vol. 46, no. 3, pp. 1–24, 2012.

[4] S. Hunold and J. L. Träff, "On the state and importance of reproducible experimental research in parallel computing," *arXiv preprint arXiv:1308.3648*, 2013.

[5] K. Hinsen, "A data and code model for reproducible research and executable papers," *Procedia Computer Science*, vol. 4, pp. 579–588, 2011.

[6] "Spectrum processing programme," https://bitbucket.org/igankevich/spec-factory.