SAND2015-0271C

The Role of Container Technology in Reproducible Computer Systems Research

Ivo Jimenez, Carlos Maltzahn (UC Santa Cruz)
(ivo|carlosm)@cs.ucsc.edu
Adam Moody, Kathryn Mohror (Lawrence Livermore National Laboratories)
(moody11|kathryn)@llnl.gov
Jay Lofstead (Sandia National Laboratories)
gflofst@sandia.gov
Remzi Arpaci-Dusseau (University of Wisconsin-Madison)
remzi@cs.wisc.edu

***Abstract*—Evaluating experimental results in the field of Computer Systems is a challenging task, mainly due to the many changes in software and hardware that computational environments go through. In this position paper, we analyze salient features of container technology that, if leveraged correctly, can help reducing the complexity of reproducing experiments in systems research. We also discuss the benefits and limitations of using containers as a way of reproducing research in other areas of experimental systems research.**

I. Introduction

A key component of the scientific method is the ability to revisit and replicate previous results. Registering information about an experiment allows scientists to interpret and understand results, as well as verifying that the experiment was performed according to acceptable procedures. Additionally, reproducibility plays a major roll in education since the amount of information that a student has to digest increases as the pace of scientific discovery accelerates. By having repeatable experiments, a student can learn by looking at provenance information, re-evaluate the questions that the original experiment answered and thus "stand in the shoulder of giants".

In applied Computer Science an experiment is carried out in its entirety by a computer. Repeating a result doesn't require a scientist to rewrite a program, rather it entails obtaining the original program and executing it (possibly in a distinct environment). Thus, in principle, a well documented experiment should be repeatable automatically (e.g. by typing `make`), however, this is not the case. Today's computational environments are complex and accounting for all possible effects of changes within and across systems is a challenging task [1,2].

Version control systems (VCS) are sometimes used to address some of these problems. By having a particular version ID for the software used for an article's experimental results, reviewers and readers can have access to the same codebase [3]. However, availability of the source code does not guarantee reproducibility [4] since the code might not compile and, even if compilable, the results might differ, in which case the differences have to be analyzed in order to corroborate the validity of the original experiment.

Additionally, reproducing experimental results when the underlying hardware environment changes challenging mainly due to the inability of predicting the effects of such changes in the outcome of an experiment. A Virtual Machine (VM) can be used to partially address this issue but the overheads associated in terms of performance (the hypervisor "tax") and management (creating, storing and transferring them) can be high and, in some fields of Computer Science such as Computer Systems research, cannot be accounted easily [5].

Container technology [6] is currently employed as a way of reducing the complexity of software deployment and portability of applications in cloud computing infrastructure. Arguably, containers have taken the role that package management tools had in the past, where they were used to control upgrades and keep track of change in the dependencies of an application [7]. In this work, we make the case for containers as a way of tackling some of the reproducibility problems in Computer Systems research. Specifically, we propose to use the resource accounting and limiting components of OS-level virtualization as a basis for creating execution profiles of experiments that can be associated with results, so that these can subsequently be analyzed when an experiment is evaluated. In order to reduce the problem space, we focus to local and distributed storage systems in various forms (e.g., local file systems, distributed file systems, key-value stores, and related data-storage engines) since this is one of the most important areas underlying cloud computing and big-data processing, as well as our area of expertise.

The rest of this paper is organized as follows. We first describe the distinct levels of reproducibility that can be associated with scientific claims in systems research and give concrete examples in the area of storage systems (section II). We then analyze salient features of container technology that are relevant in the evaluation of experiments and introduce what in our view is missing in order to make containers a useful reproducibility tool for storage systems research (section III). We then follow with a discussion about the benefits and limitations of using containers in other areas of experimental systems research (section IV). We finally discuss related work (section V) and conclude (section VI).

II. Experimental Evaluation of Computer Systems Research

An experiment in systems research is composed by a triplet of (1) workload, (2) a specific system where the workload runs and (3) results from a particular execution. Respective to this order is the complexity associated to the evaluation of an experiment: obtaining the exact same results is more difficult than just getting access to the original workload. Thus, we can define a taxonomy to characterize the reproducibility of experiments:

1. *Workload Reproducibility.* We have access to the original code and the particular workload that was used to obtain the original experimental results.
2. *System Reproducibility.* We have access to hardware and software resources that resemble the original dependencies.
3. *Results Reproducibility.* The results of the re-execution of an experiment are valid with respect to the original.

In storage systems research, workload reproducibility is achieved by getting access to the configuration of the benchmarking tool that defines the IO patterns of the experiment. For example, if an experiment uses the Flexible IO Tester[1] (FIO), then the workload is defined by the FIO input file.

System reproducibility can be divided in software and hardware. The former corresponds to the entire software stack from the firmware/kernel up to the libraries used by an experiment, whereas the latter comprises the set of hardware devices involved in the experiment such as specific the CPU model, storage drives or network cards that an experiment ran on.

Reproducing results does not necessarily imply the regeneration of the exact same measurements, instead it entails validating the results by checking how close (in shape or trends) to the original experiment they are. Given this, evaluating an experiment can be a subjective task. We propose metrics within the domain of storage systems in terms of resource utilization, specifically memory, CPU, and bandwidth, as a way of having objective metrics that do not give rise to ambiguity while comparing results.

[1]https://github.com/axboe/fio

III. Containers For Reproducible Systems Research

Current implementations of OS-level virtualization (e.g. LXC[2] or OpenVZ[3]) include an accounting component that keeps track of the resource utilization of a container over time. In general, this module can account for CPU, memory, network and IO. By periodically checking and recording these metrics while an experiment runs, we can obtain a profile of an experiment's execution. This profile is the signature of the experiment on the particular hardware that it runs on. The challenge is to recreate results on distinct hardware. By having the performance profile along with the performance profile of the underlying hardware, we can provide enough information for researches to use while evaluating a particular result. In concrete, we propose the following mapping methodology:

1. Obtain the profile of the original hardware
2. Obtain the resource utilization configuration of each container
3. Obtain the profile of the new hardware
4. Generate a configuration for the new hardware based on 2-4

The hardware profile is composed of static (e.g. the output of `lshw`) and dynamic information (e.g. the execution of micro-benchmarks to characterize the bare-metal performance of a machine).

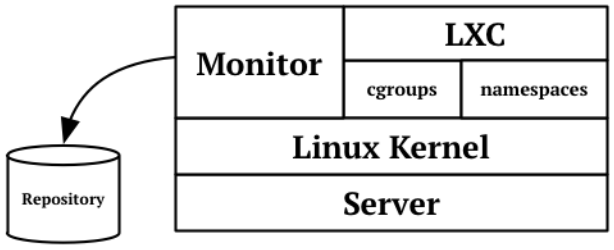

Fig. 1. A userpace process running alongside the container manager (LXC) that periodically probes the statistics of containers in order to obtain an execution profile.

Using LXC as an example, we show in Figure 1 a monitoring daemon running in as a userspace process in the host. This process periodically dumps the content of the `cgroups` pseudo-filesystem in order to capture the runtime metrics of the containers running in the system.

An alternative for structuring this information is by defining the following schema: `(IMAGE ID | EXECUTION ID | HW PROFILE ID | CGROUPS CONF | EXECUTION PROFILE)`. Where `IMAGE ID` corresponds to the image where the container was instantiated from. `EXECUTION ID` corresponds to a particular execution of the experiment with associated timestamps. `HW PROFILE ID`, as mentioned

[2]https://www.kernel.org/doc/Documentation/cgroups
[3]https://wiki.openvz.org/Proc/user_beancounters

above, captures the bare-metal specification of the machine where the container executes.

The execution database can be located remotely in a central repository that serves as the hub for managing experiments in a distributed environment. For example, this monitoring component could be implemented as a submodule of CloudLab [8]. For experiments consisting of multiple hosts and container images, orchestration tools such as Mesos [9] can also be extended to incorporate this profiling functionality.

IV. Discussion

We discuss other benefits and limitations of containerization, as well as general reproducibility guidelines when working with containers.

A. *Cataloging Experiments*

By storing profiles of experiments, we can create categories of container metrics that describe in a high-level what the experiment's goal is, for example:

- In-memory only
- Storage intensive
- Network intensive
- CPU intensive
- 50% of caching effects

Assume there is a central repository of experiments such as CloudLab [8]. A scientist wanting to test new ideas in systems research can look for experiments in this database by issuing queries with a particular category in mind.

B. *Can All Systems Research Be Containerized?*

Based on previous performance evaluations of container technology [10–13] we can extrapolate the following conditions for which experimental results will likely be affected by the implementation of the underlying OS-level virtualization:

- Memory bandwidth is of significant importance (i.e. if 5% of performance will affect results).
- External storage drives can't be used, thus having the experiment perform I/O operations within the filesystem namespace where the container is located.
- Network address translation (NAT) is required.
- Distinct experiments consolidated on the same host.
- For an experiment, containers with conflicting roles need to be co-located in the same physical host (e.g. two database instances on the same host).
- Kernel version can't be frozen.

Any experiment for which any of the above applies should be carefully examined since the effects of containerization can affect the results. The design of the experiment should explicitly account for these effects.

C. *Other Lessons Learned So Far*

We list some of the lessons that we have learned as part of our experience in implementing experiments in containers:

- Version control the experiment's code and its dependencies, leveraging git subtrees/submodules (or alike) to keep track of inter-dependencies between projects. For example, if a git repository contains the definition of a Dockerfile, make it a submodule of the main project.
- Refer to the specific version ID that a paper's results were obtained from. Git's tagging feature can also be used to point to the version that contains the codebase for an experiment (e.g. "sosp14").
- When possible, add experimental results as part of the commit that contains the codebase of an experiment. In other words, try to make the experiment as self-contained as possible, so that checking out that particular version contains all the dependencies and generated data.
- Keep a downloadable container image for the version of the experiment codebase (e.g. use the docker registry and its automated build feature).
- Whenever possible, use CI technologies to ensure that changes to the codebase don't disrupt the reproducibility of the experiment.
- Obtain a profile of the hardware used (eg. making use of tools such as SoSReport[4]), as well as the resource configuration of every container and publish these as part of the experimental results (i.e. add it to the commit that a paper's results are based on).

V. Related Work

The challenging task of evaluating experimental results in applied computer science has been long recognized [14–16]. This issue has recently received a significant amount of attention from the computational research community [1,17–21], where the focus is more on numerical reproducibility rather than performance evaluation. Similarly, efforts such as *The Recomputation Manifesto* [22] and the *Software Sustainability Institute* [23] have reproducibility as a central part of their endeavour but leave runtime performance as a secondary problem. In systems research, runtime performance *is* the subject of study, thus we need to look at it as a primary issue. By obtaining profiles of executions and making them part of the results, we allow researchers to validate experiments with performance in mind.

In [4], the authors took 613 articles published in 13 top-tier systems research conferences and found that 25% of the articles are reproducible (under their reproducibility criteria). The authors did not analyze performance. In our case, we are interested not only in being able to

[4]https://www.github.com/sosreport/sos

rebuild binaries and run them but also in evaluating the performance characteristics of the results.

Containers, and specifically docker, have been the subject of recent efforts that try to alleviate some of the reproducibility problems in data science [24]. Existing tools such as Reprozip [25] package an experiment in a container without having to initially implement it in one (i.e. automates the creation of a container from an "non-containerized" environment). Our work is complementary in the sense that we look at the conditions in which the experiment can be validated in terms of performance behavior if it runs within a container.

VI. Conclusion

In this paper we have presented our proposal for complementing container management infrastructure to capture execution profiles with the purpose of making these available to experimental research reviewers and readers. We are in the process of testing this ideas with concrete published papers in several areas of systems research.

Acknowledgements: This work was partially supported by NSF grant ... We wish to thank ...

VII. References

[1] J. Freire, P. Bonnet, and D. Shasha, "Computational reproducibility: State-of-the-art, challenges, and database research opportunities," *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, New York, NY, USA: ACM, 2012, pp. 593–596.

[2] D.L. Donoho, A. Maleki, I.U. Rahman, M. Shahram, and V. Stodden, "Reproducible research in computational harmonic analysis," *Computing in Science & Engineering*, vol. 11, Jan. 2009, pp. 8–18.

[3] C.T. Brown, "How we make our papers replicable," 2014. Available at: http://ivory.idyll.org/blog/2014-our-paper-process.html.

[4] C. Collberg, T. Proebsting, G. Moraila, A. Shankaran, Z. Shi, and A. Warren, *Measuring reproducibility in computer systems research*, Tucson, Arizona, United States: University of Arizona, 2014.

[5] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J.N. Matthews, "Xen and the art of repeated research," *Proceedings of the annual conference on USENIX annual technical conference*, Berkeley, CA, USA: USENIX Association, 2004, pp. 47–47.

[6] S. Soltesz, H. Pötzl, M.E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," *Proceedings of the 2Nd ACM SIGOPS/EuroSys european conference on computer systems 2007*, New York, NY, USA: ACM, 2007, pp. 275–287.

[7] R. Di Cosmo, S. Zacchiroli, and P. Trezentos, "Package upgrades in FOSS distributions: Details and challenges," *Proceedings of the 1st international workshop on hot topics in software upgrades*, New York, NY, USA: ACM, 2008, pp. 7:1–7:5.

[8] R. Ricci and E. Eide, "Introducing CloudLab: Scientific infrastructure for advancing cloud architecturesand applications,"*;login:* vol. 39, Dec. 2014, pp. 36–38.

[9] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," *Proceedings of the 8th USENIX conference on networked systems design and implementation*, Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308.

[10] M. Xavier, M. Neves, F. Rossi, T. Ferreto, T. Lange, and C. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," *2013 21st euromicro international conference on parallel, distributed and network-based processing (PDP)*, 2013, pp. 233–240.

[11] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," *technology*, vol. 28, 2014, p. 32.

[12] M. Xavier, M. Veiga Neves, and C. Fonticielha de Rose, "A performance comparison of container-based virtualization systems for MapReduce clusters," *2014 22nd euromicro international conference on parallel, distributed and network-based processing (PDP)*, 2014, pp. 299–306.

[13] X. Tang, Z. Zhang, M. Wang, Y. Wang, Q. Feng, and J. Han, "Performance evaluation of light-weighted virtualization for PaaS in clouds," *Algorithms and architectures for parallel processing*, X.-h. Sun, W. Qu, I. Stojmenovic, W. Zhou, Z. Li, H. Guo, G. Min, T. Yang, Y. Wu, and L. Liu, eds., Springer International Publishing, 2014, pp. 415–428.

[14] J.P. Ignizio, "On the establishment of standards for comparing algorithm performance," *Interfaces*, vol. 2, Nov. 1971, pp. 8–11.

[15] J.P. Ignizio, "Validating claims for algorithms proposed for publication," *Operations Research*, vol. 21, May. 1973, pp. 852–854.

[16] H. Crowder, R.S. Dembo, and J.M. Mulvey, "On reporting computational experiments with mathematical software," *ACM Trans. Math. Softw.*, vol. 5, Jun. 1979, pp. 193–203.

[17] J. Freire, D. Koop, E. Santos, and C. Silva, "Provenance for computational tasks: A survey," *Computing in Science Engineering*, vol. 10, May. 2008, pp. 11–21.

[18] V. Stodden, F. Leisch, and R.D. Peng, *Implementing reproducible research*, CRC Press, 2014.

[19] C. Neylon, J. Aerts, C.T. Brown, S.J. Coles, L. Hatton, D. Lemire, K.J. Millman, P. Murray-Rust, F. Perez, N. Saunders, N. Shah, A. Smith, G. Varoquaux, and E. Willighagen, "Changing computational research: The challenges ahead," *Source Code for Biology and Medicine*, vol. 7, Dec. 2012, pp. 1–2.

[20] J. Cheney, L. Chiticariu, and W.-C. Tan, "Provenance in databases: Why, how, and where," *Found. Trends databases*, vol. 1, Apr. 2009, pp. 379–474.

[21] R. LeVeqije, I. Mitchell, and V. Stodden, "Reproducible research for scientific computing: Tools and strategies for changing the culture," *Computing in Science Engineering*, vol. 14, Jul. 2012, pp. 13–17.

[22] I.P. Gent, "The recomputation manifesto," *arXiv:1304.3674 [cs]*, Apr. 2013.

[23] S. Crouch, N. Hong, S. Hettrick, M. Jackson, A. Pawlik, S. Sufi, L. Carr, D. De Roure, C. Goble, and M. Parsons, "The software sustainability institute: Changing research software attitudes and practices," *Computing in Science Engineering*, vol. 15, Nov. 2013, pp. 74–80.

[24] C. Boettiger, "An introduction to docker for reproducible research, with examples from the r environment," *arXiv:1410.0846 [cs]*, Oct. 2014.

[25] F. Chirigati, D. Shasha, and J. Freire, "ReproZip: Using provenance to support computational reproducibility," *Proceedings of the 5th USENIX conference on theory and practice of provenance*, Berkeley, CA, USA: USENIX Association, 2013, pp. 1–1.