The Role of Container Technology in Reproducible Computer Systems Research

Ivo Jimenez and Carlos Maltzahn
University of California Santa Cruz
{ivo,carlosm}@cs.ucsc.edu

Adam Moody and Kathryn Mohror
Lawrence Livermore National Laboratories
{moody11,kathryn}@llnl.gov

Jay Lofstead
Sandia National Laboratories
gflofst@sandia.gov

Remzi Arpaci-Dusseau and Andrea Arpaci-Dusseau
University of Wisconsin-Madison
{remzi,dusseau}@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 reduce the complexity of reproducing experiments in systems research. We present a use case in the area of distributed storage systems to illustrate the extensions that we envision, mainly in terms of container management infrastructure. 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 experiments. Registering information about an experiment allows scientists to interpret and understand results, as well as verify that the experiment was performed according to acceptable procedures. Additionally, reproducibility plays a major role in education since the amount of information that a student has to digest increases as the pace of scientific discovery accelerates. By having the ability to repeat experiments, a student can learn by looking at provenance information, re-evaluate the questions that the original experiment answered and thus "stand on the shoulder of giants".

In applied computer science an experiment is carried out in its entirety on a computer. Repeating an experiment 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 code base [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 that 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 is challenging mainly due to the inability to predict 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 in terms of performance (the hypervisor "tax") and management (creating, storing and transferring) can be high and, in some fields of computer science such as systems research, cannot be accounted for easily [5].

OS-level virtualization [6] is a server virtualization method where the kernel of an operating system allows for multiple isolated user space instances, instead of just one. Such instances (often called containers, virtualization engines (VE), virtual private servers (VPS), or jails) may look and feel like a real server from the point of view of its owners and users. In addition to isolation mechanisms, the kernel often provides resource management features to limit the impact of one container's activities on the other containers. Container technology 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 on 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

978-1-4799-8218-9/15 $31.00 © 2015 IEEE
DOI 10.1109/IC2E.2015.75

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 subsequently present a use case that illustrates the benefits of the proposed container management extensions that we envision (section IV). We then follow with a discussion about the benefits and limitations of using containers in other areas of experimental systems research (section V). We finally discuss related work (section VI) and conclude (section VII).

II. Evaluation of Experimental Systems Research

An experiment in systems research is composed of 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 with 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 I/O patterns of the experiment. For example, if an experiment uses the Flexible I/O Tester[1] (FIO), then the workload is defined by the FIO input file.

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

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. If our aim is to elevate our discipline to the same rank of other experimental sciences, evaluation of results should never be subjective. In systems research, result reproducibility depends on the particular goals of the experiment; within the domain of storage systems, we propose to evaluate results based on resource utilization metrics, specifically memory, CPU, and I/O bandwidth, to provide objective standards for unambiguously 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 I/O usage. By periodically checking and recording these metrics while an experiment runs, we can obtain a profile of its execution. This profile is the signature of the experiment on the particular hardware on which it executed. The challenge is to recreate results on distinct hardware. By coupling this execution profile to a profile of the underlying hardware, we can provide valuable information for researchers to use while evaluating a particular result. In concrete, when trying to reproduce an experiment on a system B that originally ran on A, we propose the following mapping methodology:

1. Obtain the hardware profile of A.
2. Obtain the configuration of every container involved in the experiment, along with their execution profile.
3. Obtain the hardware profile of B.
4. Generate a configuration for the experiment w.r.t. B by recreating the resource allocation that the experiment had when it ran on A.

Using LXC as an example, we show in Figure 1 a monitoring daemon running as a userspace process in the host that implements the process described above.

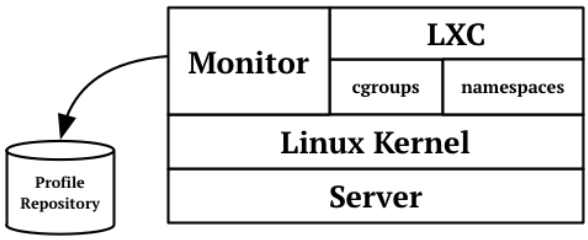

Fig. 1. A userpace process running alongside the container execution engine (LXC) that periodically probes the statistics of containers in order to obtain an execution profile.

The hardware profile is composed of static (e.g. the hardware characteristics as seen by `lshw`) and dynamic information

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

(e.g. the results of micro-benchmarks that characterize the bare-metal performance of a machine). The container configuration corresponds to the host's resources available to a container, such as number of CPUs and total amount of memory. While the experiment runs, performance metrics for the container can be obtained in order to create an execution profile.

The monitoring process periodically dumps the content of the `cgroups` pseudo-filesystem in order to capture the runtime metrics of containers running in the system. An alternative for structuring this information is by defining the following schema:

```
(IMG|EX_ID|HW_PROFILE|CGROUPS|EX_PROFILE)
```

Where `IMG` points to the image from where the container was instantiated. `EXE_ID` corresponds to a particular execution of the experiment with associated timestamps. `HW_PROFILE`, as mentioned above, captures the bare-metal performance of the machine where the container executes. `CGROUPS` is the configuration in terms of control groups for the distinct subsystems (CPU, memory, network and I/O). `EX_PROFILE` is the profile for a particular execution; i.e. there is one profile for every entry (one for each `EX_ID`).

The profile 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. One of main goals of our work is to determine whether this information is sufficient to evaluate a result.

IV. Use Case: Scalability Experiments of Ceph OSDI '06

To illustrate the utility of having execution and hardware profiles, we take the Ceph OSDI '06 paper [10] and reproduce one of its experiments. In particular, we look at the scalability experiment from the data performance section (6.1). The reason for selecting this paper is that we are familiar with these experiments. This makes it easier to reason about contextual information not necessarily available directly from the paper.

The experiments in Section 6.1 of the original paper showed the ability of Ceph to saturate disk evenly among the drives of the cluster. Figures 5-7 from the original paper showed per-OSD performance as the object size varied from 4 KB to 4 MB. Results of the scalability experiment are presented in Section 6.1.3 of the Ceph paper (Figure 8 on the original paper; reprinted below in Figure 2). The goal of this experiment is to show that Ceph scales linearly with the number of storage nodes, up to the point where the network switch is saturated. This linear scalability is our reproducibility evaluation criteria for this specific experiment.

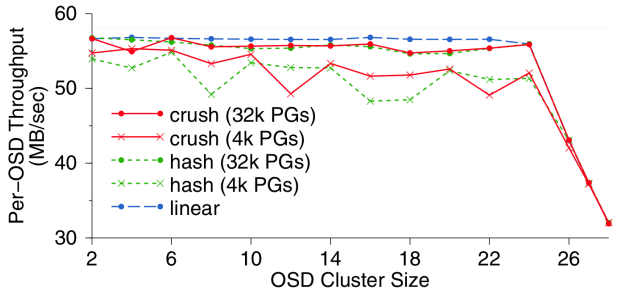

Fig. 2. Reprinting Figure 8 from the original paper. The original caption reads: "*OSD write performance scales linearly with the size of the OSD cluster until the switch is saturated at 24 OSDs. CRUSH and hash performance improves when more PGs lower variance in OSD utilization.*"

The experiment used 4 MB objects to minimize random I/O noise from the hard drives. We ignore the performance of the `hash` data distribution and increase the number of placement groups to 128 per node, thus we meaningfully compare against the red solid-dotted line in Figure 8 of the Ceph paper.

A. Reproducing Results on Similar Hardware

A subset of the hardware used for the Ceph experiments is still available in our laboratory. Each node in the system consist of a 2-core 2212 AMD Opteron @2.0GHz, 8GB of RAM, 1GbE NIC and 250GB Seagate Barracuda ES hard drives. We created a containerized version of the experiment using the 0.87 branch of Ceph. We use docker 1.3.3 and LXC 1.0.6 running on Ubuntu 12.04 hosts (3.13.0-43 x86_64 kernel).

The original scalability experiment ran with 20 clients per node on 20 nodes (400 clients total) and varied the number of OSDs from 2-26 in increments of 2. Every node was connected via 1 GbE link, so the experiment theoretical upper bound was 2GB/s (when there was enough capacity of the OSD cluster to have 20 1Gb connections) or alternatively when the connection limit of the switch was reached. The paper experiments were executed on a Netgear switch. This device has a capacity of approximately 14 GbE in *real* total traffic (from a 20 advertised), corresponding to the 24 * 58 = 1400 MB/s combined throughput shown in the original paper.

We scaled down the experiment by reducing the number of client nodes to 1 (running 16 client threads). This means that our network upper bound is approximately 110 MB/s (the capacity of the 1GbE link from the client to the switch). We throttle I/O at 30 MB/s, so this is our scaling unit (the per-OSD increment). The reason for throttling at 30 MB/s is that, over time, the Seagate disks have aged (they are 10 years old!) and overall performance among the hard drives of our cluster is different from the ~58 MB/s observed in the original paper. In order to amortize, we had to take the lowest common denominator which in this case is 30 MB/s. We throttle I/O by configuring LXC containers with the control group `blkio.throttle.write_bps_device` directive.

Figure 3 shows results of this scaled-down, throttled version of the scalability experiment. An open question is to determine if the process of scaling-down and throttling resources can be automated, given the profile repository described in the previous section.

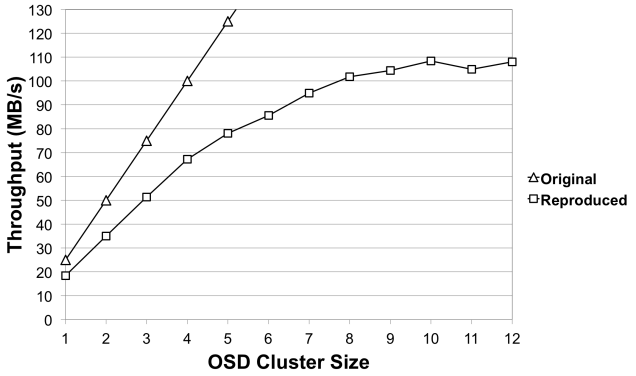

Fig. 3. Reproducing a scaled-down version of the original OSDI '06 scalability experiment. The y-axis represents average throughput, as seen by the client. They x-axis corresponds to the size of the cluster (in number of object storage devices (OSD). The square marker corresponds to the average of 10 executions. The line with triangle markers projects the original results to our setting. This projection is obtained by having the 58 MB/s divided by 2 (to reflect the doubled I/O operation of the current Ceph version), i.e. 24 MB/s as the scalability unit of the original experiment.

We see that Ceph scales linearly with the number of OSDs, up to the point where we saturate the 1GbE link[4]. We note that we don't see 30 MB/s of net I/O utilization since the current version of Ceph issues two I/O calls on each write request, one to the write-ahead log and another one to the data backend. The original experiments used a prototype version of Ceph that didn't include this atomicity/durability feature. Figure 3 also shows a projection of the original data to our setting. The original result shows better scalability behavior due to newer and more stable hard drives.

B. Reproducing Results on Different Hardware

So far we have discussed how to reproduce an experiment on the original hardware. But, as we have mentioned before, the challenge is in reproducing experiments on different hardware. Having the experiment implemented in containers allows us to swap components of the underlying hardware and repeat the experiment easily; after all, this is one of the ultimate goals of virtualization, and containers aim at doing it with minimal overhead. Our interest is in measuring the effects that replacing distinct components has on experimental results. Our conjecture is that, for many cases, the mapping methodology defined in the previous section will allow to reproduce results on distinct hardware. As part of our efforts, we are working in characterizing the cases for which our methodology will work and those for which it won't.

Thus, one of our initial goals is to empirically test the repercussions of replacing storage, CPU, memory and network devices (among others). We now present preliminary results on the outcome of swapping distinct storage drives. We re-executed the scalability experiment, swapping four old hard drives with newer models. The results are shown in Figure 4.

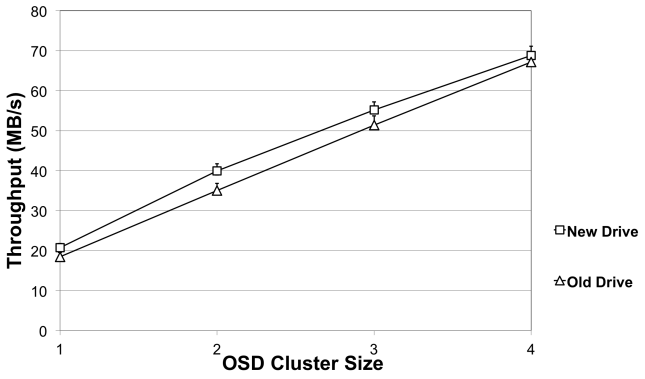

Fig. 4. Showing the effect of replacing 4 hard drives with newer models. Old hard drives are the same used in the previous figure and correspond to a set of 10 year old 250GB Seagate Barracuda ES (ST3250620NS). New hard drives correspond to 500GB Western Digital Re (WD5003ABYZ) drives. Every data point corresponds to the average (and standard error) of 10 executions.

The newer hard drives have the capacity to write at ~130 MB/s but we throttle I/O in order to replicate the behavior of our older drives. Standard error markers show that differences for two of the data points are statistically significant. Our expectation was to find complete overlapping points, since at this scalability levels (1-4), variance is relatively low. Additionally, the `blkio` cgroups subsystem has been empirically shown to effectively isolate I/O operations at low loads [11].

After investigating further about the reason of these differences, we found the following. As mentioned earlier, Ceph issues two I/O calls on each write request, one of them being asynchronous. The cgroups `blkio` controller responsible for limiting I/O on block devices (which we configure to 30 MB/s) cannot throttle asynchronous I/O operations since this type of requests go to a queue that is shared at the system level by all containers running in the host. We experientially corroborated this by executing a microbenchmark using FIO that executed the same load (4MB files) on the two hard drives in question, but using direct I/O exclusively. In this case, the performance corresponds to the throttled 30 MB/s (lines perfectly overlap). We then executed a mixed workload of both direct and async I/O requests and observed that the newer hard drive performs better than the old one, with similar results as those showed in Figure 4.

V. Discussion

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

[4]The experiment scales linearly up to 4 nodes. At OSD number 5, the 1 GbE link begins to exhibit the effects of network pressure. We empirically corroborated this by re-executing the experiment with two client nodes, in which case the experiment scales linearly up to 8 OSD nodes; at OSD number 9, the two links begin to be pressured (approximately 140 MB/s). This data is available at the repository associated to this paper (see *Section V.C*).

with containers.

A. Cataloging Experiments

By storing performance profiles of experiments and associated hardware, 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

Assuming there is a central repository of experiments associated with infrastructures 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, in particular LXC [12–15], 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 extreme importance (i.e. if 5% of performance will affect results). Some experiments show a significant overhead of the memory subsystem while handling the limits imposed to containers.
- External storage drives can't be used, thus having the experiment perform I/O operations within the filesystem namespace where the container is located. For example, AUFS has been shown to incur a penalty when processes write to the container's filesystem.
- Network address translation (NAT) is required.
- Distinct experiments consolidated on the same host. Even though many containers can be co-located in the same host, isolating and accounting can affect the performance of the server host. This can be minimized by placing as few containers as possible on the same host.
- Kernel version can't be pinned to a particular version. Since the OS is the hypervisor, distinct kernel versions observe distinct performance characteristics.

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. This list is not comprehensive, of course, and an open problem is to delineate precisely the boundaries between experiments well-suited to be reproducible via containers vs. those that are not.

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. "`osdi_14`").
- 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 continuous integration (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[5]), 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).

We followed this guidelines for this paper. All the resources needed to generate its content can be found in our github account[6].

VI. Related Work

The challenging task of evaluating experimental results in applied computer science has been long recognized [16–18]. This issue has recently received a significant amount of attention from the computational research community [1,19–21], 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

[5] https://www.github.com/sosreport/sos
[6] https://github.com/ivotron/woc_2015#camera-ready

are reproducible (under their reproducibility criteria). The authors did not analyze performance. In our case, we are interested not only in being able to 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.

VII. 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 illustrated the benefits of our hardware and container execution profiling methodology by reproducing a previously published experiment in the area of distributed storage systems. We are currently in the process of further testing these ideas on experiments from other areas of systems research such as kernel development and network systems.

VIII. 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] S.A. Weil, S.A. Brandt, E.L. Miller, D.D.E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," *Proceedings of the 7th symposium on operating systems design and implementation*, Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[11] Y. Sfakianakis, S. Mavridis, A. Papagiannis, S. Papageorgiou, M. Fountoulakis, M. Marazakis, and A. Bilas, "Vanguard: Increasing server efficiency via workload isolation in the storage i/O path," *Proceedings of the ACM symposium on cloud computing*, New York, NY, USA: ACM, 2014, pp. 19:1–19:13.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

[18] 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.

[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] 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.

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

[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.