On the Applicability of Virtualization in an Industrial HPC Environment

Tilman Küstner
Technische Universität München, Germany
Tilman.Kuestner@in.tum.de

Carsten Trinitis
Technische Universität München, Germany
Carsten.Trinitis@in.tum.de

Josef Weidendorfer
Technische Universität München, Germany
Josef.Weidendorfer@in.tum.de

Andreas Blaszczyk
ABB Corporate Research Center, Switzerland
Andreas.Blaszczyk@ch.abb.com

Patrik Kaufmann
ABB Corporate Research Center, Switzerland
Patrik.Kaufmann@ch.abb.com

Marcus Johansson
ABB Corporate Research Center, Sweden
Marcus.Johansson@se.abb.com

ABSTRACT

Virtual machines and, to a lesser extent, operating system level virtualization (also referred to as containers) allow for creation of tailored execution environments. This allows software installations to only have to target one environment, making deployment much easier for different platforms. For this reason the technique is used nowadays in most cloud systems.

In this paper we present an industrial setting in which a simulation code should be available on both different HPC cluster environments as well as on standalone workstations. To make deployment easier, we propose the use of virtualization and describe our prototype. Furthermore, we discuss benefits and challenges.

Keywords

HPC; Electric Field Simulation; Virtualization; Containers

1. INTRODUCTION

The introduction of PC-based clusters in the late 1990s has been recognized as a major step to widely enable High Performance Computing (HPC) in industry. Clusters allowed affordable parallel computing of technical applications typically performed with typically tens or hundreds of processors. The reduction of computation times from a few days or overnight to the range of minutes contributed to close integration of numerical simulations into the industrial design environment. For designer teams distributed across the world a reasonable environment has been provided by installing central HPC-clusters dedicated to specific applications which can be accessed through the corporate network. This type of operation has started at the beginning of the 2000s and successfully continues until today [4].

However, with the growing number of applications and users it turned out that the operation of dedicated cluster servers is not always economical. In some cases, where the high availability of computational resources is required and the clusters are dedicated for a single application, the utilization of hardware may be reduced to less than 5-10 %. Furthermore, due to limitations of data transfer rates across intercontinental corporate networks the dedicated servers need to be replicated in order to provide the requested compute capacity worldwide. This generates significant maintenance effort related to porting applications to different hardware platforms and operating systems. All these experiences created an evident need to rearrange the available resources in such a way that applications can be run on any hardware without maintaining a dedicated environment. In addition, the cost accounting according to usage is requested. The concept of virtualization addressed in our paper has been recognized as promising to fulfill the industrial needs.

2. BACKGROUND

The technology behind system virtualization is known since the 70-ties [11], and it is used since that time e.g. by IBM for their main frame systems up to now. However, virtualization on cheap mass-market computer systems only became available in the first years of the current century, when the increased performance of x86 systems made this approach feasible [12]. Early implementations employed workarounds for issues with x86 such as filtering the instruction stream or avoiding problematic instructions at all [1]. The new interest triggered processor vendors to implement new execution modes and hardware features for faster virtualization, reducing overheads in every new processor implementation. This guided the path to the huge market around cloud services today. By being able to efficiently virtualize at the border of hardware and software of a regular x86 system, users can bundle their software in complete system installations including their choice of an operating system and run these installations remotely. The costly maintenance and administration, including transparent backup and repair of failed components, can be outsourced to specialized hardware providers. Concentrating the hardware needs of large parts of the IT industry results in huge consolidation benefits, and competition in low prices. Private clouds, which actually are the old way for a company to run its own server hardware, only makes sense today in limited scenarios such

COSH-VisorHPC 2017 Jan 24, 2017, Stockholm, Sweden

ISBN 978-3-00-055564-0
DOI: 10.14459/2017md1344416

as with specific demands for the security of used data.

While there are a lot of benefits of virtualizing a full system, it is quite resource intensive. Often it is enough to provide an own view to a file system encapsulating a software installation as well as being able to control resource usage by means of regular operating system features. This lighter way of a virtual environment for a group of processes running on top of a given OS is called *containers*. An OS provides own name-spaces for resources such as mounted file systems, open files and network connections, or process IDs to processes inside containers. For the mainline Linux kernel, this is only available since a few years using so-called CGroups[1]. However, there is a large number software available for the administration of containers, with the most known being Docker [10]. As the added functionality for containers mostly consists of adding indirections to system calls, they do not change the performance of HPC code [13, 7].

The HPC community is quite conservative about virtualization solutions because of its eventual overheads. The additional latency introduced by virtualization layers (even in hardware) can disturb the performance of carefully tuned codes. Furthermore I/O hardware often is controlled directly from user level, bypassing the operating system for faster communication and synchronization. The latter complicates virtualization. However, the clear benefits to HPC users, reducing the dependencies to the execution environment by bundling complete software installations, are recognized by HPC computer centers. It should be clear from the above discussion that container solutions are preferred. As the complexity of controlling a lot of resources is not needed in an HPC context (such as restricting network access), software recently is developed such as Singularity[2] or PRoot[3], focusing mostly on a separate file system view.

From the users' point of view, it is convenient to be able to develop his/her HPC code in exactly the execution environment as provided by the HPC compute center, especially as workstations or even laptops run on multi-core processors nowadays, representing already a parallel environment. Furthermore, one can expect that there will be no issues around software packages traditionally provided by the compute center environment, as software dependencies are packaged within a container. One exception here are HPC libraries (such as MPI) able to directly talk to network hardware such as Infiniband, resulting in a dependence of the container to special hardware. If the user knows that performance of his/her application does not depend much on communication, s/he may decide to package a generic MPI implementation. In this case, the container could be embedded into a VM image and easily run also on cloud services, bringing additional benefits such as higher availability and scalability in exchange to spent costs for these systems. However, people regularly check cloud systems for their fitness for HPC codes, and often they fail due to the unknown hardware and connectivity between allocated compute resources [9]. Nevertheless, tailored services for HPC are created by cloud providers[4] which may getting better in the future. Since recently, cloud providers offer increased node performance by adding accelerators such as GPUs. Lower required node counts result in lower reliance on fast communication and synchronization, making such cloud systems more attractive to HPC.

[1]Committed in 2007 by Paul Menage and Rohit Seth.
[2]http://singularity.lbl.gov
[3]http://proot.me
[4]https://aws.amazon.com/de/hpc

3. APPLICATION

As a good candidate for the virtualization approach we consider a numerical program for computation of electrostatic fields based on the boundary element method (BEM) [8]. This ABB in-house program (POLOPT) has become very popular in dielectric design of power devices and is used worldwide by designers of switchgears, transformers as well as other high voltage equipment manufactured at ABB. It is well integrated with CAD-systems, enables automatic 3D-optimization [5] and can be efficiently used for evaluation of electric discharges [3], see application example in Figure 1.

The parallelization of our BEM-application is based on the de-facto message passing standard MPI [2]. According to the implemented master-slave model, the workers (slaves) perform the time consuming process of building the fully populated BEM matrix independently from each other and store the corresponding part of the matrix in local memory. An iterative GMRES solver is executed on the master node and performs parallel matrix vector multiplication by referencing the matrix parts residing on the slaves. This simple algebraic parallelization scheme turned out to be very efficient for up to 100-200 processors (cores). Within this range, the efficiency is almost independent from the structure and performance of the underlying communication network. This feature provides a good foundation for flexible virtualization.

4. APPROACH

4.1 Traditional Approach

Traditionally in high performance computing the software running the simulation (i.e. POLOPT in our case) has to be deployed and configured on each machine or compute cluster it is supposed to run on. This is the job of the system administrator, who has o adjust the users' environment accordingly such that the user can submit his or her parallel simulation run. Up until now, this has been accomplished within ABB using the Simulation Toolbox environment. The Simulation Toolbox platform has become well established in power device design [5]. However, setting up a compute cluster running the Simulation Toolbox including CAD- and numerical simulation- and optimization tools requires a non negligible administrative and maintenance effort as well as full root access to the server system.

4.2 Container/VM based Approach

Cloud computing has become a wide area of research within the last ten years. With powerful server farms becoming more and more affordable, users of high performance computing (mainly database driven[5][6]) have started to run their code in the cloud. This approach is becoming increasingly popular for simulation applications as well, e.g. CAD vendors (e.g. Autodesk[7] or PTC[8]) as well as CAE vendors like

[5]http://www.ptc.com/services/cloud/solutions
[6]http://www.ptc.com/services/cloud/solutionsapplications
[7]http://www.autodesk.com/360-cloud
[8]http://www.ptc.com/services/cloud/solutions

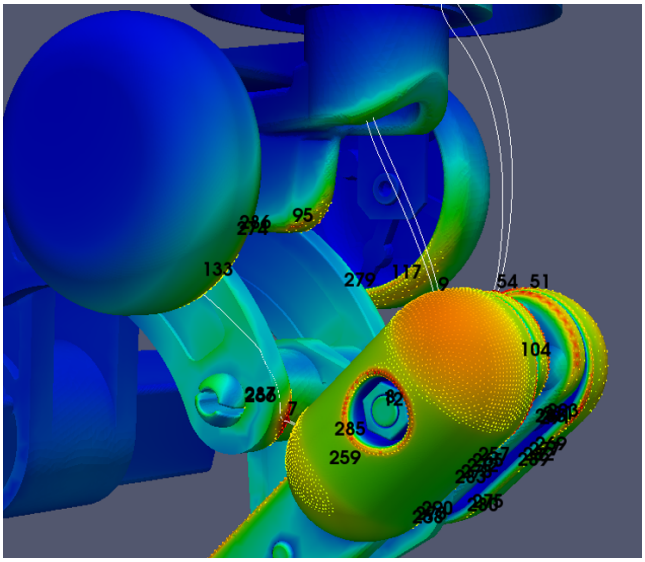

Figure 1: An example of 3D electric field computation (colors correspond to field strength) with evaluation of critical spots (denoted by black numbers) and discharge path (white trajectories) showed in Paraview-based visualizer of the ABB in-house tool Virtual High Voltage Lab [3].

e.g. SimScale[9] are offering cloud based solutions for their code. In addition, research projects aim at providing a comprehensive simulation environment within a cloud based architecture. CloudSME[10], an international research project funded by the European Union, falls into this category.

The abovementioned cloud based approaches are mainly driven by virtualization or container based technologies. Generally speaking, if communication is an issue, a container based approach should be preferred over a virtual machine base approach, as communication between virtual machines is slower than between containers. Although POLOPT's performance is not as communication critical as that of other simulation programs, we decided to use a container approach for an initial implementation. For this reason, the idea behind this paper is to port and adapt our POLOPT code to Docker technology [10]. When moving to a container based implementation, the following advantages and disadvantages need top be taken in to account:

- A container based approach has less hardware dependency than a traditional approach, i.e. the installation does not need to be reconfigured when hardware parameters change.
- A container based approach is "hardware agnostic", i.e. hardware specific features can be ignored from the users' point of view.
- A container based approach provides the well known cloud advantages such as:
  - The possibility of outsourcing the computations yielding no system administration efforts.
  - The possibility to generate specific usage statistics,
  - automatic backup,
  - increased reliability, and
  - better scalability.
- Customizable architecture, i.e. adjust resources according to requirements,
- A uniform development system for all platforms, which makes code development simple as only one version needs to be maintained.
- A container based approach might not be as efficient if the efficiency and performance of the code depend on specific hardware such as e.g. an Infiniband network.

As mentioned above, POLOPT's performance is not as dependent on the underlying network performance (POLOPT scales well even on TCP/IP based Ethernet [6]). Therefore we have decided to not use Infiniband hardware for communication and stick with TCP/IP Ethernet, which eliminated the abovementioned drawback of the container based approach.

5. PROTOTYPE

We want our prototype to be deployed in different use cases. For this reason, there actually are multiple parts to be used in the different scenarios.

Our prototype consists of the following parts:

- a container based on Debian including a POLOPT binary to be used for running as MPI task. This is compiled and linked with the Intel 17.0 compiler and Intel MPI. Thus, the container includes corresponding libraries. The TCP transport back-end is used for communication.
- a few adaptation scripts for our target HPC cluster environment, able to call a given job scheduler.
- a container with a HTTP server, a web page, as well as an Intel MPI installation with the mpirun command. The web page reachable via the HTTP server provides a portal page to (1) request the execution of a POLOPT solver run for given input data and (2) provides feedback to the user on currently executing solver runs including a progress report. For this, the container checks for two different scenarios. Either it runs on the login node, and has access to the adaptation scripts, or it is to be run on a standalone workstation. For the first, a CGI-script calls the adaptation scripts outside of the container which generates and submits a corresponding job script, first exporting the MPI installation outside of the container. For the latter, the compute container is launched first, and afterwards mpirun is called directly.

The scenario for the deployment on a regular HPC cluster is given in Figure 2, showing the different parts of our prototype described above. In contrast, Figure 3 presents the scenario where a user wants to run POLOPT on his/her

[9] http://simscale.com/
[10] http://cloudsme.eu/

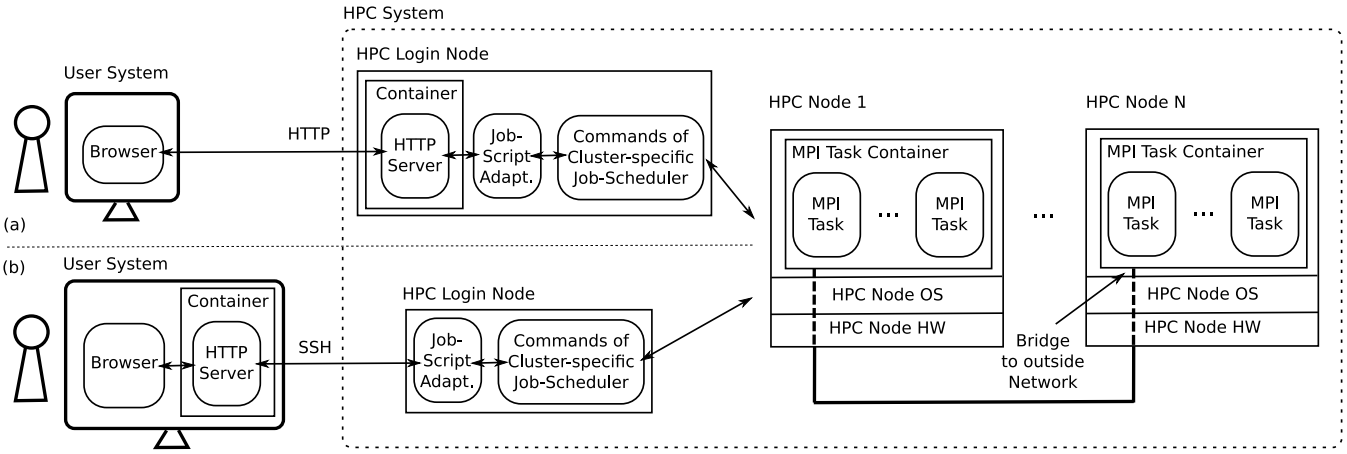

Figure 2: An HPC code using MPI tasks within containers on multiple nodes of an HPC cluster installation with a regular job scheduler. Depending on the HPC cluster provider, a front-end server starting and observing jobs may either run on the login node (a) or has to be installed on the user system (b). In either case, (small) adaption scripts to the job scheduler software used in the cluster have to be provided.

own workstation. This uses the system VM software Virtualbox[11] to be able to run on e.g. Windows or MacOS hosts.

We note here the challenges which had to be solved during development:

- We want MPI code using containers embedding the MPI tasks to be run on multiple nodes. For the MPI tasks to see each other, we decided to use bridged network interfaces to be able to access processes running inside of the containers on the various nodes, see Fig. 2.
- We had to make sure that the topology of compute nodes is passed through to the inside of our containers, as the MPI environment sees the container configuration e.g. regarding available CPUs and NUMA nodes. This is needed as MPI pins MPI tasks to logical CPUs, and any container configuration should keep the configured fixed binding.

First measurements using Docker containers have shown no performance loss on a single node. The sequential run showed exactly the same performance with and without Docker, which is important when comparing scalability, since slower sequential implementations are likely to improve scalability. Since this is not the case here, we ran the same relatively small input model (approx. 7100 unknowns) on a four core single node. In the final version of this paper, comprehensive measurements will be provided together with exact runtime numbers.
We expect the container based version to perform at full speed: For the reasons mentioned in section 4.2 POLOPT should scale well even with a TCP/IP based network as being used for our Docker based approach.

6. FIRST MEASUREMENTS

For the following first results, we run a small model with around 7100 unknowns, using one to four cores in different scenarios:

- direct execution.
- execution in Docker containers, with one container per MPI task[12]. Numbers shown are with containers already launched.
- execution within the system-VM VMware Workstation, using VTX. For the measurements, a guest VM configured with 4 CPU cores is running (with 1 GB memory).

The setup consists of a single system with an Intel Core i5-3450 quad-core processor (Ivy Bridge) with 3.1 GHz nominal clock rate, 6 MB L3 cache, and 8 GB main memory. This processor has support for hardware virtualization (Intel VTX with EPT). The operating system used on the host (and for the container scenario) is Ubuntu 16.04.01. As container solution, Docker 1.12.5 is used. As system-VM, VMware Workstation 12 Player with guest OS Debian 8.6 is used. POLOPT was compiled with the Intel compiler (ifort/icc 17.0.0) and Intel MPI 2017 was used.

Fig. 4 shows the absolute runtimes for the abovementioned scenarios. Results are averaged over 10 runs, respectively. To better understand scalability issues, Fig. 5 presents speedup numbers.

As can be seen, runtimes are quite similar. The VM solution is a little bit slower due to virtualization overhead. However, the speedup using the container approach is slightly worse than for both the native and system-VM approach. Further investigation of this phenomenon will be subject to future work.

7. DISCUSSION

Implementing the approach described above, we anticipate our simulation code to show the same performance with and without a container. Networking should not be a big concern, as POLOPT scales well on both Infiniband and

[11] `https://www.virtualbox.org`

[12] Another possibility is to have one container for all MPI tasks running on a compute node. This may be better for communication and is under investigation.

Ethernet. In general, simulation software which is not heavily reliant on a high performance network should suffer little slowdown in a virtualized environment. We do not expect a present job scheduling system to have specific extensions to run containers; instead, we provide adaptation scripts which allow to initialize the containers when a job is started and cleaning up afterwards.

In the case of using a cloud environment, there are both advantages and disadvantages. Most important is data security as CAD models and other data required by the simulation might be important business secrets. Data can be encrypted for transport on public networks but needs to be decrypted when running the actual computation. While newer processors have ISA extensions to limit access from OS- to user-level (e.g. enrypted data in DDR memory), such hardware deployed in a cloud does not naturally improve the consumer's confidence in the cloud provider.

Further, the customer has limited control over hardware, updates and monitoring of resources. Performance might be less predictable. If the simulation code depends heavily on inter-node communication, a high performance network is required. Also, for large amount of input/output data, a high performance parallel file system is desirable.

Using a cloud environment for simulation codes offers several advantages. Model data can be shared and versioned among engineers using a storage cloud. We hope to see higher flexibility and scalability as the same code (inside a container) can be be run on an engineer's laptop computer as well as a high performance cluster. The hardware on which the container starts can be chosen based on the simulation code and input data. For example, accelerator-aware codes can be launched on compute nodes with GPUs. Larger input models which require more main memory would start several containers on several nodes.

In a best case scenario, an industrial company does not need to buy and manage dedicated hardware. Flexible payment models, as offered by cloud hosting providers, also reduce costs. Better utilization of hardware capacity, better hosting and reduced energy consumption make this scenario also more environmentally friendly.

Deployment of new software and software updates is simplified. The workload of a system administrator is also reduced, as libraries and other prerequisites need not be installed on the system. Instead, they come packaged inside the container. This also reduces time and effort of end-users (engineers). They need less knowledge about accessing the compute cluster or the job scheduling system. Instead, they find a lean and clear web front-end, which shows number and progress of running jobs. Optionally, costs per job could be displayed to the end-user as well, in order to make the accounting more transparent.

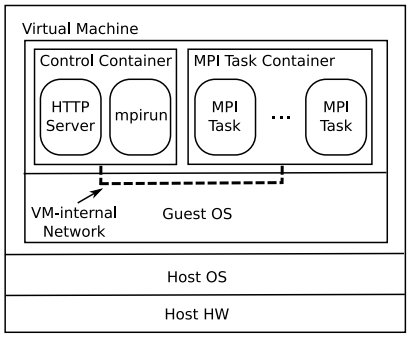

Figure 3: On a standalone workstation, the same container images as in the HPC cluster scenario can be used. Running them inside a VM allows independence from the OS of the workstation.

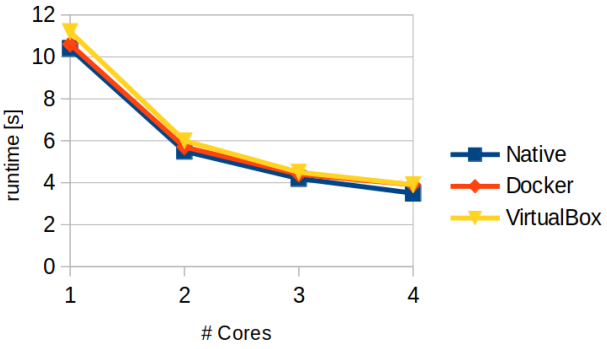

Figure 4: Absolute runtimes of a small POLOPT model in various scenarios, running on 1 up to 4 cores (MPI only).

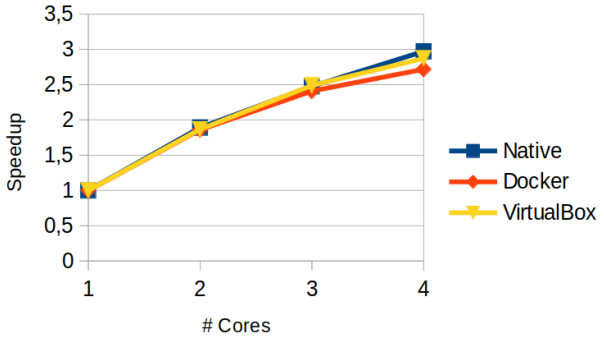

Figure 5: Speedups of a small POLOPT model in various scenarios, running on 1 up to 4 cores (MPI only).

8. CONCLUSIONS AND OUTLOOK

In this paper, we propose to enable the benefits of virtualization for industrial HPC codes. That is, simulation codes are deployed within containers and available for execution in different runtime environments such as HPC clusters and workstations. First measurements, using a relevant simulation code within ABB, show that packaging industrial codes in this way provides the end-user with reliable and fast computational resources even at peak times.

For future work, we will extend our measurements both to multiple nodes and other applications. For the latter, it is especially important to also look into more communication intensive applications.

If an industrial use case can risk to use cloud computing, we can assume that all execution environments use system VMs. In this case, approaches such as library-OSes which link applications and required OS functionality become interesting, as this can improve efficiency (no context switches between guest OS and guest user level). We will look into

such novel approaches in the future.

Acknowledgment

The authors would like to thank Anton Schreck for providing the prototype setup and first measurements.

9. REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, Oct. 2003.

[2] A. Blaszczyk, Z. Andjelic, P. Levin, and A. Ustundag. Parallel computation of electric fields in a heterogeneous workstation cluster. In *High-Performance Computing and Networking Conference*, volume 919 of *LNCS*, pages 606–611. Springer, 1995.

[3] A. Blaszczyk, J. Ekeberg, S. Pancheshnyi, and M. Saxegaard. Virtual high voltage lab. In *Scientific Computing in Electrical Engineering (SCEE) Conference*, Strobl, Austria, Oct 2016.

[4] A. Blaszczyk, H. Karandikar, and G. Palli. Net value! Low cost, high-performance computing via the Intranet. *ABB Review*, (1):35–42, 2002.

[5] A. Blaszczyk, J. Ostrowski, B. Samul, and D. Szary. Simulation Toolbox. In *ABB Review*, volume 3, 2013.

[6] A. Blaszczyk and C. Trinitis. Experience with PVM in an industrial environment. In *European Parallel Virtual Machine Conference*, pages 174–179. Springer, 1996.

[7] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*, pages 171–172. IEEE, 2015.

[8] N. D. Kock, M. Mendik, Z. Andjelic, and A. Blaszczyk. Application of 3-d boundary element method in the design of ehv gis components. *IEEE Electrical Insulation Magazine*, 14(3):17–22, May/Jun 1998.

[9] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas. Performance evaluation of amazon ec2 for nasa hpc applications. In *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date*, ScienceCloud '12, pages 41–50, New York, NY, USA, 2012. ACM.

[10] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), Mar. 2014.

[11] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[12] M. Rosenblum. The reincarnation of virtual machines. *ACM Queue*, 2(5):34–40, 2004.

[13] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 233–240. IEEE, 2013.