Vol. 18, n.° 4, pp. 31-42, 2019
Revista UIS Ingenierías
Página de la revista: revistas.uis.edu.co/index.php/revistauisingenierias
Universidad Industrial de Santander

Performance evaluation of container-based virtualization for high performance computing environments

Evaluación de rendimiento de virtualización basada en contenedores en ambientes de computación de alto rendimiento

Carlos Arango[1a], Rémy Dernat[2b], John Sanabria[1c]

[1] EISC, Facultad de Ingeniería, Universidad del Valle, Cali, Colombia.
Email: [a] carlos.arango.gutierrez@correounivalle.edu.co, [b] remy.dernat@umontpellier.fr,
[c] john.sanabria@correounivalle.edu.co
[2] ISEM, CNRS, Univ. Montpellier, IRD, EPHE, Montpellier France
Email: john.sanabria@correounivalle.edu.co

Received: 23 February 2019. Acepted: 14 September 2019. Final version: 16 October 2019.

Abstract

Virtualization technologies have evolved along with the development of computational environments. Virtualization offered needed features at that time such as isolation, accountability, resource allocation, resource fair sharing and so on. Novel processor technologies bring to commodity computers the possibility to emulate diverse environments where a wide range of computational scenarios can be run. Along with processors evolution, developers have implemented dierent virtualization mechanisms exhibiting enhanced performance from previous virtualized environments. Recently, operating system-based virtualization technologies captured the attention of communities abroad because their important improvements on performance area. In this paper, the features of three container-based operating systems virtualization tools (LXC, Docker and Singularity) are presented. LXC, Docker, Singularity and bare metal are put under test through a customized single node HPL-Benchmark and a MPI-based application for the multi node testbed. Also the disk I/O performance, Memory (RAM) performance, Network bandwidth and GPU performance are tested for the COS technologies vs bare metal. Preliminary results and conclusions around them are presented and discussed.

Keywords: Container-based virtualization; linux containers; singularity; docker; high performance computing.

Resumen

Las tecnologías de virtualizacíon han evolucionado a la par con el desarrollo de los ambientes computacionales ofreciendo características como aislamiento, contabilidad, asignación de recursos y el compartir recursos de forma justa; entre otros. Las nuevas generaciones de procesadores incluyen primitivas de virtualización que permiten emular diversos ambientes de computación. Junto con la evolución de los procesadores, los desarrolladres han implementando diferentes mecanismos de virtualización que mejoran el rendimiento de anteriores herramientas de virtualización. Recientemente, la virtualización a nivel del sistema operativo ha atraido la atención de usuarios de la computación en general debido a las mejoras exhibidas en el rendimiento. En este artículo se ponen a prueba cuatro ambientes: LXC, Docker, Singularity y *bare metal*. Usando diversos *benchmarks*, para un solo nodo y aplicaciones basadas en MPI sobre múltiples nodos, se probaron diferentes subsistemas como: E/S, memoria RAM, tráfico de red y GPU. Resultados preliminares y sus conclusiones son presentados y discutidos.

Palabras clave: Container-based virtualization; Linux containers; Singularity; Docker; High performance computing.

ISSN Printed: 1657 – 4583, Online: 2145 – 8456, **CC BY-ND 4.0**
How to cite: C. Arango, R. Dernat, J. Sanabria, "Performance evaluation of container-based virtualization for high performance computing environments," *Rev. UIS Ing.*, vol.18, no. 4, pp. 31-42, 2019. doi: 10.18273/revuin.v18n4-2019003

1. Introduction

Computational tools are key elements in the development of dierents areas of knowledge such as industry, research and academy. Simulations and modeling are important computational techniques used to reduce waiting times and money budgets bringing novel and eective solutions to challenging problems.

New solutions usually required to be obtained through processor-intensive applications which demand specialized infrastructures to perform on acceptable time. High Performance Computing (HPC) is the name given to those processor-intensive applications to take advantage of massive parallel infrastructures known as computational clusters.

Computational clusters fulfill most of the processor-intensive applications requirements, tackling novel problems and presenting foreseeable solutions. However, more challenging problems surpass the capacity of one computational cluster and federations of scattered clusters are necessary to meet the needs of these problems. Those federations of clusters are known as Grid systems.

Grid systems oer virtual organizations which integrate users and computational resources abroad. Thus, multiple virtual organizations are consolidated world wide tackling diverse problems (e.g. cancer cure, search for fundamental particles and sequencing genomes, among others) then requiring diverse services and applications. This babel of tools presents a challenging problem for system administrators who have to deal with library versions, dependencies and software compatibility.

Virtualization is not a new technology [1] but it has been recently reactivated because of the advantages that it exhibits. Nowadays, o the shelf processors incorporate optimized virtualization instructions to support the deployment of secure and isolated computational environments bringing power ecient computational environments able to run several services in one single box [2,3]. Cloud computing then emerges as a new infrastructure to borrow the best of Grid Computing and Virtualization in such a way that several users and projects are able to share computational resources in an isolated fashion, [4]. Cloud computing additionally exhibits other characteristics such as ubiquitous access, scalability on-demand and pay for consumed resources, [5]. Infrastructure, development platforms and software services have took advantage of it and a new economy around to Cloud computing infrastructures have emerged [6].

However, HPC is one of the few scenarios where Cloud computing has fall short on providing the performance expected by HPC applications. Although important milestones have been reached in the virtualization context and some cloud providers make available tailored virtual

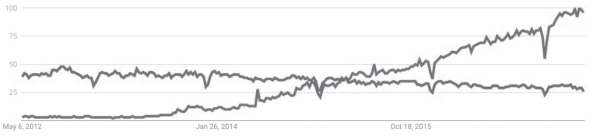

Figure 1. Container (blue) vs Virtual machines (red) interest over time. [8]

computational tools, the performance of virtualized contexts are very slow when they are compared with their bare metal counterpart [7].

Many scientific and academic applications taking advantage of native and optimized processor instructions which are penalized when they are executed on top of hypervisor tools. Hypervisors present a simplified view of the native hardware to the virtual machines then they can barely access to the optimized set of instructions of actual processors.

An alternative approach to the hypervisor-based solution to virtualized environments has gained traction and attention. Containers[9] subtract the hypervisor layer of the virtualization equations and relies on namespaces and cgroups in order to provide isolation and accounting of the consumed resources by the container instances.

Then, the rapid development of container-based technologies is getting attention of Internet users because containers accelerates the development process, eases distribution and deployment of applications, Figure 1. Leaders of such development are Docker[1] [10] and Linux Containers (LXC [11]). Nevertheless its implications for scientific computing including HPC are still on doubt. Containers are proving to be an extremely valuable technology for science delivering portability and reproducibility to the users. Containers can provide the requirements of a program and execute it directly, without the overhead that comes with hypervisor-based approaches. “Singularity-containers” from [12] is a container-based approach which focuses on providing portable environments which could leverage the migration of computational science to the cloud. Singularity integrates seamlessly with existing workload managers such as Slurm, HTCondor or Torque; fact that could ease its adoption of HPC facilities.

[1]http://www.docker.com

At the distributed systems and networks laboratory, at Universidad del Valle, we are working on the deployment of container-based software infrastructures to support the research process on dierent areas of knowledge. We have tested diverse operating system-based virtualization technologies running single node and multi-node applications getting important results which show that this kind of virtualization is prime time ready to support research processes. This paper presents a set of benchmarks that stress dierent aspects such as compute, memory bandwidth, memory latency, network bandwidth, and I/O bandwidth.

We will present and compare three container-based operating systems (Docker, LXC and Singularity) in section II. Then, we will describe the methodology used alongside the results in order to evaluate performance overhead of container-based technologies versus the bare metal in section III. Related works will be addressed in section IV.

2. Container-based operating system virtualization technologies

Containers are software components to enable the execution of applications on isolated environments. Containerbased operating systems (COS), also known as lightweight virtual machines, provide isolation of system resources (file system, network communications) in such a way that every container has its own set of processes ids, user identifiers, filesystem namespace and so on. Containers have a closer access to operating system services than their counterparts virtualization tools such as native virtualization, paravirtualization and hypervisors. Figure 2-a shows that containerized applications run almost at the same level of native applications. In contrast, classical virtualization approaches (Figure 2-b) propose several layers between applications in virtualized environments and the hardware where virtual machines are actually running. In fact, these layers impose a big overhead in virtualized applications when they are compared with applications running on top of bare metal systems. Therefore COS technologies are now very attractive not only because they provide experimental reproducibility and platform portability but also because they exhibit a performance close to the performance exhibited on top of native environments [13].

COS have being around for awhile and there are numerous implementations of it. On 2000, FreeBSD (4.0) featured the Jails system which focused on providing an isolated filesystem (an enhanced version of the chroot command). Solaris goes a step further with its operating system OpenSolaris providing not only isolation services but also mechanisms related to snapshots and cloning. These aforementioned projects were mostly supported by BSD operating systems. On 2005 OpenVZ was announced as a COS implementation for Linux systems. Despite it was an open source project there was not too much interest in the Linux community then it was barely included into the Kernel main stream. OpenVZ never gets enough track amongst Linux community.

LXC (Linux Containers) took advantage of the namespace concept. Dierent from previous approaches where file system isolation was provided, LXC extended the isolation property to users, processes and networking. On 2001, Linux supported the first file system namespace known as the mount namespace. Since then, other namespaces have been supported, UTS, IPC, PID, user and network namespaces. In addition to isolation, on 2006, Google project (process containers) implemented a functionality to limit the resource usage, e.g. CPU, memory, disk I/O, network). This project was later merged into the Linux kernel and it was named cgroups (control groups). From that, cgroups capabilities have been extended to firewalling and unified hierarchy, amongst others.

Docker released on 2013, was basically an additional layer on top of LXC exposing additional features such as mounted storage, network port redirection, and container catalog management. These features made Docker a prime time product for the industry.

Singularity is a project developed at Lawrence Berkeley National Laboratory (LBNL) and it is mainly focused on experimental reproducibility and isolation.

2.1. LXC

LXC is built on top of kernel namespaces which is a Linux kernel feature that isolates and virtualizes system resources such as processes, network, filesystems, network stack, among others[2]. These capabilities allow a fully operational container-based environments exhibiting interesting features such as exposure of network services from containers, containers live migration, and a complete set of accountability mechanisms[3]. In the networking context, LXC supports route- and bridge-based networking which allow the communication with the outside world but these features add a virtual network layer over the host which imposes an overhead to the

[2]http://haifux.org/lectures/320/netLec8_final.pdf

network performance. Mechanisms based on cgroups are used for restraining the amount of resources that a container can consume, e.g. CPU, memory, number of opened files and so on. The container scheduling follows two level CPU scheduler which tries to promote fair scheduling among containers. First level scheduler determines which container will run, the second level determines which process in that container actually will run. For I/O bandwidth there is also a two-level scheduling mechanism known as Completely Fair Queuing (CFQ) scheduler. Each container has a priority and inside of it an I/O bandwidth is given according to priorities.

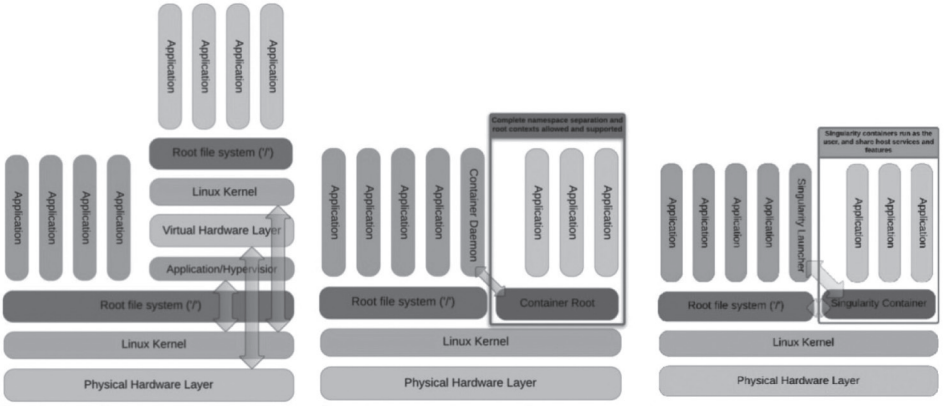

Figure 2. (a) Architecture of hypervisor-based virtual environments. (b) Architecture of container-based virtual environments. (c) Singularity Architecture.

2.2. Docker

Docker basically extends LXC with a kernel-and application level API [14] and mainly focusing on network service virtualization. Through the libcontainer library, Docker provides access to virtualization facilities provided by the Linux kernel along with some abstracted virtualized interfaces such as libvirt[3], LXC and systemdnspawn[4]. The control over host's resources is provided through Control Groups (cgroups) thus it limits the amount of resources used by a container such as memory, disk space and I/O [15]. Docker features a layered filesystem called AuFS (Advanced Multi-Layered Unification Filesystem) which allows to overlay one or more existing filesystems. When a process needs to create a copy, AuFS creates a copy of that file. This feature provides image versioning management and exposing base images to more specialized virtualized systems [10]. Docker has emerged as a key player in the virtualization field since it has being widely adopted in the industry and academy because it leverages infrastructure consolidation and exhibits a low resource footprint. Docker has boosted the adoption of service oriented architectures (e.g. microservices [16]) because it ease the deployment of selfcontained modules which are able to independently interact with third parties using well-known and widely adopted network protocols (e.g. web services). These service oriented architectures encourage the adoption of adaptable and extensible computational environments (e.g. workflows) accelerating the pace of scientific progress [17].

[3]https://libvirt.org/

[4]https://www.freedesktop.org/software/systemd/man/systemdnspawn.html

2.3. Singularity

Singularity[12] is another container-based approach developed at Lawrence Berkeley National Laboratory (LBNL) [18]. It was created with the idea of compute mobility in mind. Although Singularity uses namespaces, it is used basically for application portability instead of host virtualization. In other words Singularity virtualizes only what is necessary to achieve run-time application container and portable environments. Singularity does not support user escalation or context changes therefore Singularity's container inherits permissions of the user who runs that container. Because it does not support context change then I/O operations flow directly between environments where those operations are happening reducing the operation overhead and execution times. Singularity seamlessly integrates with diverse HPC environments and tools, e.g. resource managers, HPC file systems,

GPUs, etc. Singularity's design enables the utilization of vintage Container OS like RHEL 5 and also supports Docker-based images.

2.4. Comparing LXC, Docker and Singularity

Three COS technologies have been discussed. Table 1 presents a summary of some features described above (versions compared are given in table 2). The authors would like to note that this table, and the following discussion, is relevant to the time of writing of this paper. Docker has most of all those characteristics (e.g. user escalation, API for developers, versioning management), then it is a very handy tool for leveraging the development of sophisticated enterprise and research tools. LXC is known as the predecessor of Docker. LXC has evolved in the meantime. One big evolution is LXD which is an API for LXC. For instance, it allows the live migration of containers between dierent LXD hosts. Therefore, users of LXC/LXD are not the same than the docker users (e.g. LXC is provided by the Proxmox virtualization server solution[5]). Indeed, state full containers and full environment (like singularity) could be quite interesting features. On the other hand, Singularity exhibits a limited number of properties because it is mostly conceived for code mobility and high availability of resources. In next section, HPC benchmarks were run against these COS technologies and preliminary results exhibit that the absence of some features positively aects the performance of these benchmarks.

Table 1. Features of LXC, Docker and Singularity.

Feature	COS		
	LXC	Docker	Singularity
Support namespaces	Yes	Yes	Yes
Support cgroups	Yes	Yes	No
Support port mapping	Yes	Yes	No
User escalation	Yes	Yes	No
Unprivileged hardware access	No	No	Yes
API for applications and developers	Yes	Yes	No
Image Layering	No	Yes	Yes
Support snapshots	Yes	Yes	No([6])
Network interface	Host or Bridge	Bridge	Host
Default filesystem	Host([7])	AuFS	ext3
Access to host filesystem	Yes	Yes	Yes
Root daemon	Yes	Yes	No
Registry/Repository for the images	Yes	Yes	Yes
Build a container from a file	No	Yes	Yes
HPC accommodations	No	No	Yes
Keep modifications after restart	Yes	No	Yes

3. Methodology and benchmarks

This section studies the computational performance of COS technologies vs bare metal. We performed several experiments with the current most popular COS implementations. Virtualization technologies and their versions are given in Table 2.

The performance experiments were executed at two facilities Universidad del Valle Cluster and Montpellier Bioinformatics Biodiversity cluster computing platform. Configuration of computational nodes used on this work are as follows: CPU model Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz(64-core node); Memory 164 GB DDR3-1,866 MHz, 72-bit wide bus at 14.9 GB/s on P244br and a HPE Dynamic Smart Array B140i Disk; OS Ubuntu 16.04 (64-bit) distribution was installed on the host machine.

[5]https://www.proxmox.com
[6]However, a singularity image is only one file.
[7]Tight integration with ZFS.

We used the industry reference HPL-Lapack benchmark to test CPU performance, and microbenchmarks to individually measure memory, network, I/O and GPU overhead.

We know that results may vary significantly depending on the CPU architecture. versions of the kernel may introduce gains and losses of performance that would influence the results of experiments. Hence, we took care of compiling the same sha1sum binary for all benchmarks, using the host network for Singularity.

Table 2. Virtualization technologies and their versions.

Virtualization technologies	Versions
Singularity	2.2.1
Docker	17.03.0-ce, build 60ccb22
LXC	2.0.9

3.1. Time to execute a basic operation

The basic operation includes the start-up of the container and the execution of the very basic and very well known command "/bin/echo HelloWorld".We used "/usr/bin/time" from the host to monitor it. From a native point of view, it always took 0.00 second. The three containers (LXC, Docker, Singularity) are minimalists and have been similarly built. The images are already present on the host. We compared 6 operations fig 3 : the native "/bin/echo Hello World", and the same within "singularity exec", "docker run", "docker exec", "lxc start + lxc exec" and "lxc exec". The "docker run" command includes the boot, the execution and the shutdown of the container, while "lxc exec" and "docker exec" need a running container. So, we decided to add "lxc start + lxc exec + lxc stop" to the chart. Considering this graph, all the shutdown operations of a container are the slowest.

Then, we analyzed the strace[8] outputs of the previous commands, and we did not notice any specific bottleneck. However, we observed a significant amount of "futex" (143) and "rt_sigprocmask" (122) operations with "docker run". These operations are usually dealing with synchronization mechanisms over threads when they are accessing shared resources e.g. shared memory regions. "docker run" is an operation to create a new container (a.k.a. new running process) which requires to access and modify shared resources and data structures at kernel level.

[8]http://man7.org/linux/man-pages/man1/strace.1.html

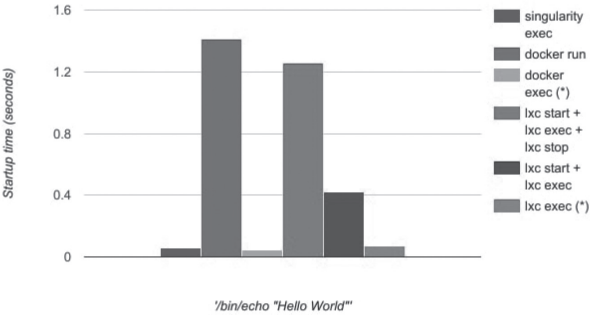

Figure 3. Elapsed time (in seconds) for running /bin/echo Hello Word".

3.2. CPU performance

In order to evaluate COS technologies for HPC we run the HPL-Benchmark [19] for a real vs virtual cluster as the ratio between the HPL benchmark performance of the cluster and the performance of a real environment formed with only one instance of same type, expressed as a percentage.

The benchmark were compiled using GNU C/C++ 5.4 and OpenMPI 2.0.2.We did not use any additional architectureor instance-dependent optimizations. We used the SHA-1 hashes [20] with the sha1sum program and checked the libraries with the ldd utility to ensure the binaries integrity. For the HPL benchmark, the performance results depend on two main factors: the Basic Linear Algebra Subprogram (BLAS) [21] library, and the problem size. We used in our experiments the GotoBLAS library, which is one of the best portable solutions, freely available to scientists. Searching for the problem size that can deliver peak performance is extensive; instead, we used the same problem size 10 times (10 N, 115840 Ns) for performance analysis.

Figure 4 shows the performance of HPL-Benchmark. The Y axis is demonstrating the dierences in technologies (that is why it doesn't goto zero). The LXC was not able to achieve native performance presenting an average overhead of 7.76 %, Docker overhead was 2.89 %, this could be probably caused by the default CPU use restrictions set on the daemon which by default each container is allowed to use a node's CPU for a predefined amount of time. Singularity was able to achieve a better performance than native with 5.42%because is not emulating a full hardware level virtualization (only the mount namespace) paradigm and as the image itself is only a single metadata lookup this can yield in very high performance benefits.

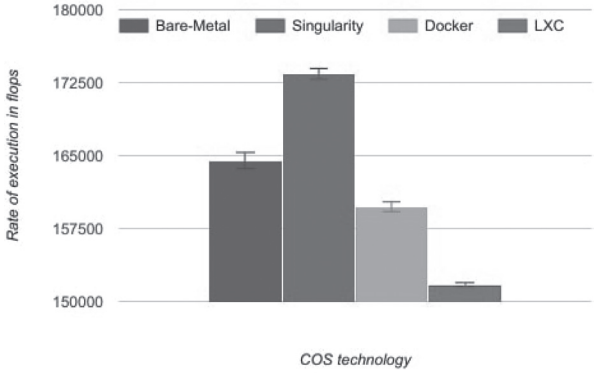

Figure 4. Rate of execution for solving the linear system.

3.3. Disk I/O performance

The disk performance was evaluated with the IOzone benchmark [22]. It generates and measures a variety of file operations and access patterns (such as Initial Write, Read, Re-Read and Rewrite).We ran the benchmark with a file size of 15GB and 64KB for the record size, under two(2) scenarios. The first scenario was a totally contained filesystem (without any bind or mount volume), and the second scenario was a NFS binding from the local cluster.

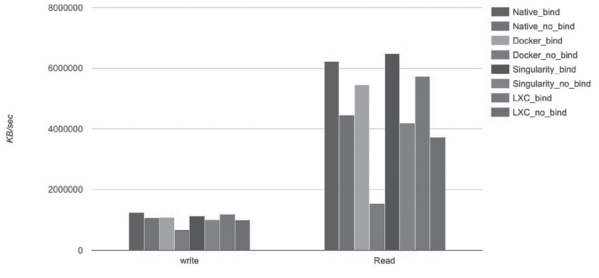

Figure 5. IOzone benchmark write and read.

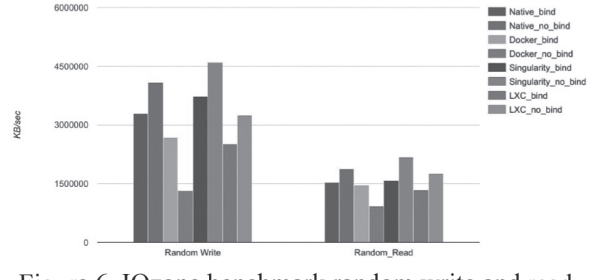

Figure 6. IOzone benchmark random write and read.

A closer inspection in COS shown in Figure 5 reveals that both LXC and Singularity had similar results for write operations. For read operations, where the Singularity slightly reach the native performance, and LXC had an overhead of 16.39% against native. On the other hand, with Docker, we observed a lost of performance of 37.28% on write and 65.25% on read. Figure 6 shows the performance of random read and random write. We noticed a similar behavior than the read and write standard operations. Docker introduces a greater overhead on random I/O processes. While LXC and Singularity filesystem implementations allows a better I/O performance, Docker advanced multi-layered unification filesystem (AUFS) has it drawbacks. When an application running in a container needs to write a single new value to a file on a AUFS, it must copy on write up the file from the underlying image. The AUFS storage driver searches each image layer for the file. The search order is from top to bottom. When it is found, the entire file is copied up to the container's top writable layer. From there, it can be opened and modified.[23]

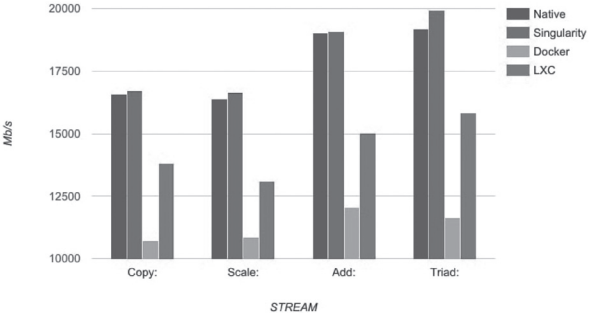

Figure 7. STREAM benchmark results.

3.4. Memory performance

The Memory performance on single node was evaluated with the STREAM application benchmark[24]. It is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels [25]. The STREAM benchmark is specifically designed to work with datasets much larger than the available cache on any given system, so that the results are (presumably) more indicative of the performance of very large, vector style applications. Performance evaluation is tight to the memory bandwidth of the system. Performance is therefore gated by memory bandwidth and not latency. The benchmark has four components: COPY, SCALE, ADD and TRIAD.

Results are presented in fig 7.

Figure 7 presents dierent performance for COS and native systems, for vector operation. This is due to the fact that container-based systems have no resource constraints and can use as much of a given resource as

the host's kernel scheduler will allow. The worst results were observed in Docker, which presented an average overhead of approximately 36% when compared to the native throughput.

3.5. Network bandwidth and latency performance

For the MPI-level network evaluation we used the MVAPICH OSU Micro-Benchmarks 5.3.2 [26] using a direct 10 Gbps Ethernet link between the nodes. We run point to point tests for measuring bandwidth and network latency.
Docker and LXC attaches all containers on the host to a bridge and connects the bridge to the network via NAT.

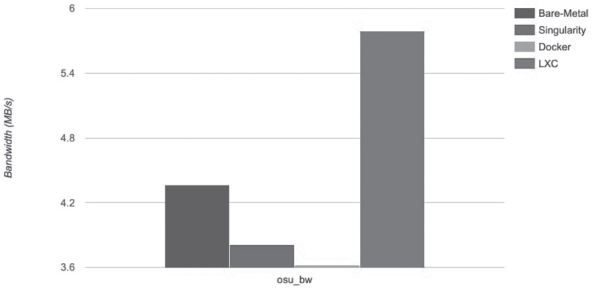

Figure 8. OSU MPI bandwidth Test msgsize 4 MB.

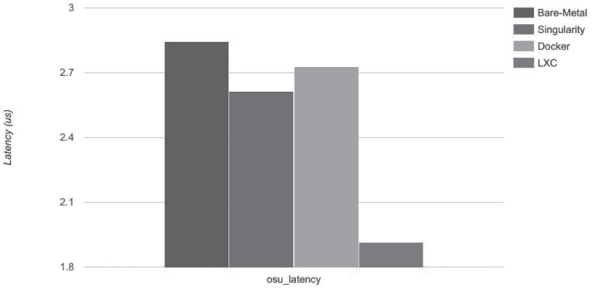

Figure 9. OSU MPI Latency Test msgsize 1 byte.

We did not set any special network configuration for any technology, more than their native networking documented on each project web page (creating bridges for LXC and using Docker Swarm and creating an overlay network for Docker). Singularity needed no additional network configurations.

Figure 8 shows the network bandwidth comparison for the COS. LXC has the best network scores with a great dierence against the other two COS being evaluated. The singularity container showed a lower performance than the native implementation followed by Docker which presented the worst results. Its average bandwidth was 16.96% smaller than native.

Figure 9 shows that LXC has less than 32% the network latency against native. The worst bandwidth and latency was observed with Docker. These results can be explained due to dierent implementations of the network isolation of the virtualization systems. While Singularity container does not implement virtualized network devices, both Docker and LXC implement network namespace that provides an entire network subsystem. COS network performance degradation is caused by the extra complexity of transmit and receive packets (e.g. Daemon processes).

3.6. GPU performance

The performance studies were executed on a Dell PowerEdge R720, with 2*Intel(R) Xeon(R) CPU E5-2603 @ 1.80GHz (8 cores) and a NVIDIA Tesla K20M.[9]. From a system point of view, we used Ubuntu 16.04.2 (64-bit), with NVIDIA cuda 8.0[28] and the NVIDIA driver version 375.26. The virtualization technologies and their versions are given in Table 3.

Table 3. Virtualization technologies and their versions.

Virtualization technologies	Versions
Singularity	2.2.1
Docker	17.03.0-ce, build 60ccb22
LXC	2.0.9

In order to evaluate COS technologies for GPU-HPC, we used the NAMD (NAnoscale Molecular Dynamics) [29] program, as a benchmark tool. We ran those GPU benchmarks on a Tesla K20m with "NAMD x86_64 multicore CUDA version 2017-03-16" [on the stmv dataset (1066628 Atoms)], using the 8 cores and the GPU card, without any specific additional configuration, except the use of the "gpu4singularity"[10] code for Singularity and the "nvidia-docker"[11] tool for Docker. For a real vs virtual cluster, the ratio is printed in the log as "days/ns"(loweris better).

Figure 10 shows the performance of NAMD-Benchmark. The Y axis is in "days/ns" (the lower the better). LXC was not able to achieve native performance. Docker achieved a better performance than native, which can be explained on the work that Nvidia is doing to build cloudnative gpu applications. Nevertheless, Docker does not natively support NVIDIA GPUs with

[9]Kepler architecture[27], GK110 Graphics processors, 2496 CORES, 208 TMUS, 40 Rops, 5120 MB Memory size, GDDR5 Memory type, 320 bit Bus width
[10]https://github.com/NIH-HPC/gpu4singularity
[11]https://github.com/NVIDIA/nvidia-docker

containers [30]. Singularity was able to achieve a better performance than native given that it provides native gpu support [12].

3.7. Source Code

The scripts to run the experiments from this paper are available at https://github.com/ArangoGutierrez/containers-benchs

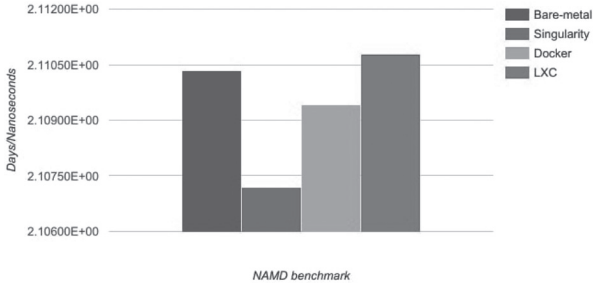

Figure 10. Tesla K20m benchmarks on NAMD.

4. Related work

Some papers have explored the overhead of containerbased virtualization tools as presented by [31, 3, 32]. Mostly they compare the performance overhead of COS versus classic Virtual machine technologies (e.g KVM, LinuxV-server). They all agree that the current resource management implementation for LXC and Docker, lead to poor isolation and security.

Containers are proving to be an extremely valuable technology for scientific research, delivering benefits such as portability and reproducibility to scientific users. COS can emulate a single program and can be executed directly, with less overhead that with running a virtual machines. Indeed, some works already described COS technologies in a scientific use case [33, 34, 35, 36]. Despite the advantages oered by container technologies, the implications for scientific computing, including HPC, are still unclear, although there are already some initiatives like Singularity, Shifter[37] (See also benchmarks on Cray systems for Shifter [38]), Charliecloud[39], cHPC[40] or Docker Universe Applications in HTCondor[12].

Many core technologies (a.k.a. GPU cards) are widely deployed on private data centers and more recently they have been exposed as services in the cloud computing landscape in order to attend the increasing growth for computational power in dierent fields of research, e.g. bioinformatics [41], storage processing [42] and multimedia [43], among others. Virtualization technologies and their impact on GPU technologies has not been broadly studied because the challenges exhibited to virtualize the functionality of GPUs and to support GPU passthrough. J.P. Walters et al.[44] run non-standardized benchmarks for KVM, Xen, VMWare and LXC. Using GPU Passthrough, these virtualization technologies are put under test running CUDA and OpenCL applications. Preliminary results exhibit a penalty over 10% on Xen and KVM. VMWare exhibited an irregular performance and LXC showed a closest performance to the native case.

To the best of our knowledge, there are no similar publications to this one. In particular, our work assesses three COS technologies using standardized benchmarks. This approach gives a preliminary approach to characterize the assessment of COS and virtualization technologies in general.

There is a little research eort around container-based solutions for heavy HPC applications. Performance evaluations on literature usually does not put under test the HPL-LAPACK benchmark, instead a version of HPLLINPACK, where matrix size could impact on the final result (e.g CPU cycles). Reports like [31] used a compiled version of LINPACK from INTEL, here we compiled the binary inside of each COS, to replicate a normal work flow, when running on a HPC cluster. Moreover, HPL results may vary significantly depending on the CPU architecture.

5. Conclusions

This paper presented a performance comparison of containerbased virtualization tools (Docker, LXC, Singularity) against bare-metal. According to our results, we observed that Singularity containers are usually more suitable for HPC implementation than Docker or LXC. From a network point of view, LXC is very ecient, however, not all namespaces are equal, and Singularity does swap out the user namespaces. Therefore, if the container have more ecient libraries than the host, the Singularity solution could yield a performance increase, while LXC and Docker control their resources by cgroups namespace, which results on a overhead for CPU intensive processes. Besides, Singularity optimizes HPC-specific libraries like CUDA or OpenMPI. For GPU applications, we recommend the implementation of Docker and Singularity to deploy

[12]http://research.cs.wisc.edu/htcondor/manual/v8.4/2_12Docker_Universe.html

on HPC clusters, or in the cloud. CUDA accelerated machine learning projects have already started offering Dockerfiles in order to run those applications over a container ready system. [30]

For I/O-intensive workloads, Container images can be much more optimal then running against shared storage (even when the container image exists on that remote storage). That is normal, as this is the same principle as cache or even the HPC scratch, that is to say a way to have data close to the process. Moreover, we would avoid the use of the standard Docker-based solutions (AUFS), due to overhead issues.

Concerning small tasks like "Hello World", or even a more consistent memory job (see STREAM results), one more time, we would avoid the use of Docker, except if the container is already running on the host. That could lead to a real problem in a HPC system where the image needs to be downloaded everywhere and then, started, before being executed. Contrary to Singularity, where you have only one file, which can be shared through the network or can be stored on a distributed filesystem. Furthermore, a distributed filesystem can be significantly impacted by the metadatas accesses, thus Singularity can limit that problem.

Singularity blocks privilege escalation within the container to avoid users of having root access. Docker instead must be isolated thus will preclude access to high performance networks (e.g. InfiniBand) and optimized storage platforms.

Considering our container-based results, COS and particularly Singularity are a good alternative to overcome the virtualization overhead issues. COS environments can be used on mixed environments where HPC and HPSS are required. In order to take the HPC scenario, COS technologies must focus on: container overhead, container technology and architecture concerns (e.g. privilege escalation and network/file system access), and workflow compatibility.

Acknowledgments

This work tests where heavily run on the Engineering faculty Cluster, University of Valle. This work also largely benefited from the Montpellier Bioinformatics Biodiversity cluster computing platform. This work was funded by COLCIENCIAS project BPIN-2013000100007 (Ci-BioFi).We would also like to thank David Godlove from the NIH Biowulf Cluster for its code on GPU.

References

[1] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: Current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[2] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[3] 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 *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. IEEE, 2013, pp. 233–240.

[4] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*. Ieee, 2008, pp. 5–13.

[5] P. M. Mell and T. Grance, "Sp 800-145. the nist definition of cloud computing," Gaithersburg, MD, United States, Tech. Rep., 2011.

[6] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee, 2008, pp. 1–10.

[7] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 159–168. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2010.69.

[8] "Google google trends," https://trends.google.com/trends/, accessed: 2017-03-15.

[9] "LinuxContainers lxc linux containers," https://linuxcontainers.org/, accessed: 2017-03-15.

[10] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[11] M. Helsley, "Lxc: Linux container tools," I*BM devloperWorks Technical Library*, p. 11, 2009.

[12] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLOS ONE*, vol. 12, no. 5, pp. 1–20, 05 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0177459.

[13] C. Ruiz, E. Jeanvoine, and L. Nussbaum, "Performance evaluation of containers for HPC," in *VHPC - 10th Workshop on Virtualization in High-Performance Cloud Computing*, ser. VHPC - 10th Workshop on Virtualization in High-Performance Cloud Computing, Vienna, Austria, Aug. 2015, p. 12. [Online]. Available: https://hal.inria.fr/hal-01195549.

[14] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[15] Wikipedia, "Docker (software) — wikipedia, the free encyclopedia," 2017, [Online; accessed 18-March-2017]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=770287241.

[16] M. Fowler and J. Lewis, "Microservices," *ThoughtWorks. http://martinfowler. com/articles/microservices. html [last accessed on February 17, 2015]*, 2014.

[17] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examiningthe challenges of scientific workflows," *Computer*, vol. 40, no. 12, 2007.

[18] "A container for hpc," http://www.admin-magazine.com/HPC/Articles/Singularity-A-Container-for-HPC, accessed: 2017-03-18.

[19] A. Petitet, "Hpl-a portable implementation of the highperformance linpack benchmark for distributed-memory computers," *http://www.netlib-. org/-benchmark/hpl/*, 2004.

[20] D. Eastlake 3rd and P. Jones, "Us secure hash algorithm 1 (sha1)," Tech. Rep., 2001.

[21] J. Dongarra, "Preface: basic linear algebra subprograms technical (blast) forum standard," *International Journal of High Performance Computing Applications*, vol. 16, no. 2, pp. 115–115, 2002.

[22] W. D. Norcott, "Iozone web page," 2012.

[23] "Docker and aufs in practice," Apr 2017. [Online]. Available: https://docs.docker.com/engine/userguide/storagedriver/aufs-driver/

[24] J. D. McCalpin, "Sustainable memory bandwidth in current high performance computers," *Silicon Graphics Inc*, 1995.

[25] ——, "Stream: Sustainable memory bandwidth in high performance computers," University of Virginia, Charlottesville, Virginia, Tech. Rep., 1991-2007, a continually updated technical report. http://www.cs.virginia.edu/stream/. [Online]. Available: http://www.cs.virginia.edu/stream/

[26] O. Micro-Benchmarks, "Osu network-based computing laboratory," *URL: http://mvapich. cse. ohio-state. edu/benchmarks*.

[27] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *IEEE micro*, vol. 28, no. 2, 2008.

[28] D. Kirk *et al*., "Nvidia cuda software and gpu parallel computing architecture," in *ISMM*, vol. 7, 2007, pp. 103–104.

[29] L. V. Kale, A. Bhatele, E. J. Bohm, and J. C. Phillips, "Namd (nanoscale molecular dynamics)," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 1249–1254.

[30] "Nvidia docker: Gpu server application deployment made easy," Feb 2017. [Online]. Available: https://devblogs.nvidia.com/parallelforall/nvidia-docker-gpu-server-application-deployment-made-easy/

[31] 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*. IEEE, 2015, pp. 171–172.

[32] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Generation Computer Systems*, vol. 68, pp. 175–182, 2017.

[33] F. Moreews, O. Sallou, H. Ménager *et al*., "Bioshadock: a community driven bioinformatics shared docker-based tools registry," *F1000Research*, vol. 4, 2015.

[34] P. Belmann, J. Dröge, A. Bremges, A. C. McHardy, A. Sczyrba, and M. D. Barton, "Bioboxes: standardised containers for interchangeable bioinformatics software," *Gigascience*, vol. 4, no. 1, p. 47, 2015.

[35] B. D. O'Connor, D. Yuen, V. Chung, A. G. Duncan, X. K. Liu, J. Patricia, B. Paten, L. Stein, and V. Ferretti, "The dockstore: enabling modular, community-focused sharing of docker-based genomics tools and workflows," *F1000Research*, vol. 6, 2017.

[36] P. Di Tommaso, M. Chatzou, E.W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature Biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[37] D. M. Jacobsen and R. S. Canon, "Contain this, unleashing docker for hpc," *Proceedings of the Cray User Group*, 2015.

[38] D. Bahls, "Evaluating shifter for hpc applications," in *Cray User Group Conference Proceedings*, 2016.

[39] R. Priedhorsky and T. Randles, "Charliecloud: Unprivileged containers for user-defined software stacks in hpc," Los Alamos National Laboratory (LANL), Tech. Rep., 2016.

[40] O. Weidner, M. Atkinson, A. Barker, and R. F. Vicente, "Rethinking hpc platforms: Challenges, opportunities and recommendations," arXiv preprint arXiv:1702.05513, 2017.

[41] S.-T. Lee, C.-Y. Lin, and C. L. Hung, "Gpu-based cloud service for smith-waterman algorithm using frequency distance filtration scheme," *BioMed research international*, vol. 2013, 2013.

[42] W. Sun, R. Ricci, and M. L. Curry, "Gpustore: Harnessing gpu computing for storage systems in the os kernel," in *Proceedings of the 5th Annual International Systems and Storage Conference*, ser. SYSTOR '12. New York, NY, USA: ACM, 2012, pp. 9:1–9:12. [Online]. Available: http://doi.acm.org/10.1145/2367589.2367595.

[43] W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia cloud computing," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 59–69, 2011.

[44] J. P. Walters, A. J. Younge, D. I. Kang, K. T. Yao, M. Kang, S. P. Crago, and G. C. Fox, "Gpu passthrough performance: A comparison of kvm, xen, vmware esxi, and lxc for cuda and opencl applications," in *2014 IEEE 7th International Conference on Cloud Computing*, June 2014, pp. 636–643.