Performance Evaluation of Container-based Virtualization for High Performance Computing Environments

Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, Cesar A. F. De Rose
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, Brazil
{*miguel.xavier, marcelo.neves*}*@acad.pucrs.br*

Abstract—**The use of virtualization technologies in high performance computing (HPC) environments has traditionally been avoided due to their inherent performance overhead. However, with the rise of container-based virtualization implementations, such as Linux VServer, OpenVZ and Linux Containers (LXC), it is possible to obtain a very low overhead leading to near-native performance. In this work, we conducted a number of experiments in order to perform an in-depth performance evaluation of container-based virtualization for HPC. We also evaluated the trade-off between performance and isolation in container-based virtualization systems and compared them with Xen, which is a representative of the traditional hypervisor-based virtualization systems used today.**

Keywords-**Container-based virtualization; Linux containers; High performance computing.**

I. Introduction

Virtualization technologies have become very popular in recent years, bringing forth several software solutions (such as Xen [1], VMware [2] and KVM [3]) and the incorporation of hardware support in commodity processors (such as Intel-VT [4] and AMD-V [5]). The main benefits of virtualization include hardware independence, availability, isolation and security. It is widely used in server consolidation and can be considered one of the foundations of cloud computing.

Nevertheless, despite its benefits, the use of virtualization has been traditionally avoided in most HPC facilities because of its inherent performance overhead [6]. There have been many studies on the performance of virtualization in the literature, a few of them focusing on HPC environments [7], [6]. In general, past studies have shown that the traditional hypervisor-based virtualization (such as Xen [1], VMware [2] and KVM [3]) has a high performance overhead, specially in terms of I/O, making prohibitive its use in HPC.

Recent operating container-based virtualization implementations (such as Linux-VServer [8], OpenVZ [9] and Linux Containers (LXC) [10]) offer a lightweight virtualization layer, which promises a near-native performance. In this context, we argue that container-based virtualization can be a powerful technology for HPC environments and propose a performance and isolation evaluation of recent container-based implementations. To support our claim, we would like to illustrate two usage scenarios:

- **Better resource sharing:** HPC clusters usually have their resources controlled by a Resource Management System (RMS), such as PBS/TORQUE [11], which enables sharing of the resources among multiple users. With the proliferation of multicore technology, current HPC clusters' nodes are composed of dozens of processing units. Since one of the primary goals of a RMS is trying to maximize the overall utilization of the system, a single multicore node can be shared by many different users. However, without an isolation layer, there are no guarantees that applications from different users will work together in the same node. In this scenario, the use of container-based virtualization could improve the resource sharing allowing for multiple isolated user-space instances. This is the case with MESOS [12], a platform that uses LXC for sharing a cluster between multiple diverse cluster computing frameworks, such as Hadoop and MPI.
- **Custom environments:** HPC clusters are typically shared among many users or institutes, which may have different requirements in terms of software packages and configurations. Even when the users share some software packages, it is hard to update them without disturbing each other. In practice, software packages in working clusters are often once installed and kept unchanged for a very long time except for some bug fix, security enhancement, or small upgrades [13]. This usage scenario makes it difficult to deploy newly developed or experimental technologies in traditional cluster environments. Hence, the use of a virtualization layer could facilitate the creation and maintenance of multiple environments customized according to the users's needs.

In both scenarios, there is a need for an isolation layer without loss of performance. Therefore, we conducted experiments using the NAS Parallel Benchmarks (NPB) [14], which is a well-known benchmark in HPC, to evaluate the performance overhead and the Isolation Benchmark Suite (IBS) [15] to evaluate the isolation in terms of performance and security. Since our focus is also on partitioning the resources of HPC clusters with multicore nodes, we evaluate the performance in both single and multi node environments, using respectively OpenMP and MPI implementation of NPB.

II. Container-based Virtualization

Resource virtualization consists of using an intermediate software layer on top of an underlying system in order to provide abstractions of multiple virtual resources. In general, the virtualized resources are called virtual machines (VM) and can be seen as isolated execution contexts. There are a variety of virtualization techniques. Today, one of the most popular is the hypervisor-based virtualization, which has Xen, VMware and KVM as its main representatives.

The hypervisor-based virtualization, in its most common form (hosted virtualization), consists of a virtual machine monitor (VMM) on top of a host OS that provides a full abstraction of VM. In this case, each VM has its own operating system that executes completely isolated from the others. This allows, for instance, the execution of multiple different operating systems on a single host.

A lightweight alternative to the hypervisors is the container-based virtualization, also known as Operating System Level virtualization. This kind of virtualization partitions the physical machines resources, creating multiple isolated user-space instances. Figure 1 shows the difference between container-based and hypervisor-based virtualization. As can be seem, while hypervisor-based virtualization provides abstraction for full guest OS's (one per virtual machine), container-based virtualization works at the operation system level, providing abstractions directly for the guest processes. In practice, hypervisors work at the hardware abstraction level and containers at the system call/ABI layer.

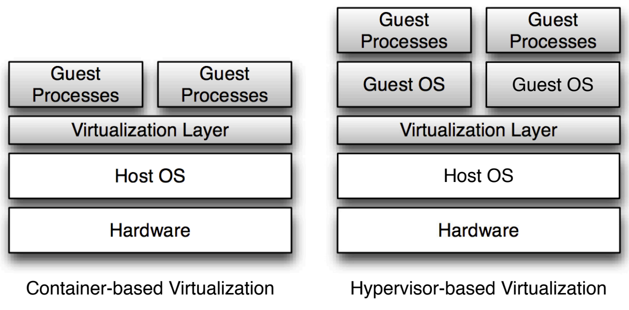


Figure 1. Comparison of container-based and hypervisor-based virtualization.

Since the container-based virtualization works at the operating system level, all virtual instances share a single operating system kernel. For this reason, container-based virtualization is supposed to have a weaker isolation when compared to hypervisor-based virtualization. However, from the point of view of the users, each container looks and executes exactly like a stand-alone OS [9].

The isolation in container-based virtualization is normally done by kernel namespaces [16]. It is a feature of the Linux kernel that allows different processes to have a different view on the system. Since containers should not be able to interact with things outside, many global resources are wrapped in a layer of namespace that provides the illusion that the container is its own system. As examples of resources that can be isolated through namespaces, consider filesystem, process IDs (PID), inter-process communication (IPC) and network [16].

On the other hand, the resources management in container-based virtualization systems is normally done by Control Groups (cgroup) [17], which restricts the resource usage per process groups. For example, using cgroups it is possible to limit/prioritize CPU, memory and I/O usage for different containers. In some cases, some systems use their own implementations to perform the resource management due to the incompatibility with cgroups.

The rest of this section presents each of the container-based virtualization systems studied in this work: Linux-VServer, OpenVZ and LXC.

A. *Linux-VServer*

Linux-VServer is one of the oldest implementation of Linux container-based system. Instead of using namespaces to guarantee isolation, Linux-VServer introduced (through a patch) its own capabilities in the Linux kernel, such as process isolation, network isolation and CPU isolation. Linux-VServer uses the traditional chroot system call to jail the file system inside the containers. That way, it limits the scope of the file system for the processes. The processes isolation is accomplished through a global PID space, which hides all processes outside of a container's scope and prohibits unwanted communications between processes of different containers. The main benefits of this approach is its scalability for a large number of containers. However, the drawback is that the system is unable to implement usual virtualization techniques, such as live migration, checkpoint and resume, due the impossibility to re-instantiate processes with the same PID [8].

Linux-VServer does not virtualizes network subsystems. Rather, all networking subsystem (such as routing tables and IP tables) are shared among all containers. In order to avoid having one container receive or sniff traffic belonging to other containers, this approach sets a container identifier tag into the packets and incorporates the appropriate filters in the networking stack to ensure that only the right container can receive them. The drawbacks is that the containers are unable to bind sockets to a subset of host IPs and to change their own route table and IP tables rules, it needs to be made by the host administrator [8].

To perform CPU isolation, Linux-VServer uses the standard Linux scheduler, overlapped by the Token Bucket Filter (TBF) [18] scheme. A token bucket is associated with each container and serves to accumulate tokens at a certain rate. In that way, every process running in a container is linked to the creation of a token. The processes of a particular container are removed from the run-queue until their bucket accumulates a certain minimum number of tokens. This token bucket scheme can be used to provide a fair sharing and work-conservation of the CPU and may also impose rigid upper limits [8]. The token bucket scheme is very similar to Xen Credit Scheduler [19].

Resource limits, such as memory consumption, number of processes and file-handles, are performed using system calls (rlimit tool) provided by the Linux kernel. In addition, the Linux-VServer kernel includes even more capabilities for limiting another types of resources, such as the number of sockets and file descriptors opened. However, the recent versions of Linux-VServer includes support to cgroups, which can also be used to restrict the CPU usage and memory consumption for containers. The Linux-VServer containers are managed by the util-vserver tools package [8].

B. OpenVZ

OpenVZ offers similar functionality to Linux-VServer. However, it is built on top of kernel namespaces, making sure that every container has its own isolated subset of a resource. The system uses a PID namespace to guarantee the process isolation between different containers. It is so that every container processes has its own unique process IDs. Furthermore, unlike Linux-VServer, the PID namespace makes possible the use of usual virtualization techniques, such as live migration, checkpoint and resume. In OpenVZ, each container has its own shared memory segments, semaphores, and messages, due the IPC kernel namespace capability. Moreover, the OpenVZ also uses the network namespace. In this way, each container has its own network stack, which includes network devices, routing tables, firewall rules and so on. It also provides some network operation modes, such as Route-based, Bridge-based and Real Network based. The main differences between them is the layer of operation. While Route-based works in Layer 3 (network layer), Bridge-based works in Layer 2 (data link layer) and Real Network in Layer 1 (physical layer). In the Real Network mode, the host system administrator can assign a real network device (such as eth1) into a container, similar to Linux-VServer, providing the better network performance [9].

OpenVZ introduces four resource management components named as User Beancounters (UBS), fair CPU scheduling, Disk Quotas and I/O scheduling. The first provides a set of limits and guarantees controlled per-container done through control parameters. In this way, we can restrict memory usage and various in-kernel objects such as IPC shared memory segments and network buffers. The OpenVZ CPU scheduler is implemented in two levels, trying to promote a fair scheduling among containers. The first level decides which container will get attention from the processor at some instant of time. The second level performs the scheduling of internal processes of the container based on priority scheduling policies, such as in Linux. There is another approach named VCPU Affinity, which tells the kernel the maximum number of CPUs that a container can use. The Disk Quota is a feature that allows to set up standard UNIX per-user and per-group disk limits for containers [9]. Finally, a similar approach of CPU scheduling is used for I/O. In this case, the second level scheduling uses Completely Fair Queuing (CFQ) Scheduler [20]. For each container is given an I/O priority, and the scheduler distributes the I/O bandwidth available according to priorities. In this way, no single container can saturate a channel, interfering with performance isolation. The OpenVZ containers are controled by the vzctl tool [9].

C. LXC

In the same way as OpenVZ, LXC uses kernel namespaces to provide resource isolation among all containers. During the container startup, by default, PIDs, IPCs and mount points are virtualized and isolated through the PID namespace, IPC namespace and file system namespace, respectively. In order to communicate with the outside world and to allow the network isolation, LXC uses the network namespaces. Two configuration are offered by LXC in order to configure the network namespaces: Route-based and Bridge-based. Unlike Linux-VServer and OpenVZ, the resource management is only allowed via cgroups. Thus, LXC uses cgroups to define the configuration of network namespaces [10]. The process control is also accomplished by cgroups, which has function of limiting the CPU usage and isolating containers and processes. I/O operations are controlled by CFQ scheduler, as in OpenVZ. LXC containers are managed by the lxc-tool [10].

III. Experiments

This section studies the performance and isolation of container-based and hypervisor-based virtualization. We performed several experiments with the current linux container-based virtualization implementations: Linux VServer, OpenVZ and LXC. We chose Xen as the representative of hypervisor-based virtualization, because it is considered one of the most mature and efficient implementations of this kind of virtualization [21].

Our experimental setup consists of four identical Dell PowerEdge R610 with two 2.27GHz Intel Xeon E5520 processors (with 8 cores each), 8M of L3 cache per core, 16GB of RAM and one NetXtreme II BCM5709 Gigabit Ethernet adapter. All nodes are inter-connected by a Dell PowerConnect 5548 Ethernet switch. The Ubuntu 10.04 LTS (Lucid Lynx) Linux distribution was installed on all host machines and the default configurations were maintained, except for the kernel and packages that were compiled in order to satisfy the virtualization systems' requirements. We know that different versions of the kernel may introduce gains or losses of performance that would influence the experiments results. Hence, we took care of compiling the same kernel version for all systems. We chose the kernel version 2.6.32-28, because it has support to all systems' patches and configurations. Therefore, for OpenVZ, we patched the kernel (2.6.32-feoktistov) and installed the package vzctl (3.0.23-8), which is necessary to manage the OpenVZ containers. We have compiled the OpenVZ kernel with the official configuration file (.config) suggested by the

OpenVZ developing team [9], in order to ensure that all OpenVZ kernel options were enabled. For Linux-VServer, we also patched the kernel (2.3.0.36.29.4) and installed the package util-vserver (0.30.216 r2842-2) to control the Linux-VServer containers. The LXC already has a mainline implementation in the official kernel source. In that way, we just installed the LXC toolkit (0.6.5-1) and ensure that all requirements present by lxc-checkconfig tool were met. Finally, for Xen, we compiled and installed the kernel (xen-4.1.2) and tools provided by the Xen package.

The rest of this section presents the results of our evaluation of all linux container-based systems.

A. Computing Performance

To evaluate the computing performance on a single computer node, we selected the LINPACK benchmark [22]. It consists of a set of Fortran subroutines that analyzes and solves linear equations by the least squares method. The LINPACK benchmark runs over a single processor and its results can be used to estimate the performance of a computer in the execution of CPU-intensive programs. We ran LINPACK for matrices of order 3000 in all container-based system and compare them with Xen. As shown in Figure 2(a), all container-based system obtained performance results similar to native (there is no statistically significant difference between the results). We believe that it is due to the fact that there are no influence of the different CPU schedulers when a single CPU-intensive process is run in a single processor. The results also show that Xen was not able to achieve the same performance, presenting a average overhead of 4.3%.

B. Memory Performance

The memory performance on a single node was evaluated with STREAM [23], a simple synthetic benchmark program that measures sustainable memory bandwidth. It performs four type of vector operations (Add, Copy, Scale and Triad), using datasets much larger than the cache memory available in the computing environment, which reduces the waiting time for cache misses and avoid memory reuse.

The results are shown in Figure 2(b). As can observed, container-based and native systems present similar performance, regardless of the vector operation. This is due to the fact that container-based systems have the ability to return unused memory to the host and other containers, enabling better use of memory. The worst results were observed in Xen, which presented an average overhead of approximately 31% when compared to the native throughput. This overhead is caused by the hypervisor-based virtualization layer that performs memory accesses translation, resulting in loss of performance. Also, Xen shows the problem of double-cache, i.e., the same blocks are used by the host and the virtual machine.

C. Disk Performance

The disk performance was evaluated with the IOzone benchmark [24]. 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 10GB and 4KB of record size.

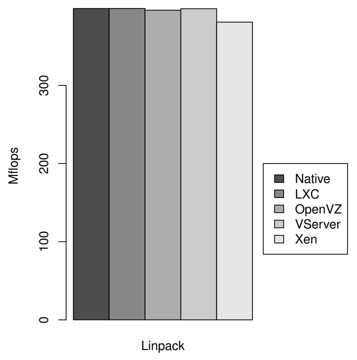
Closer inspection in container-based systems shown in Figure 2(c) revels that both LXC and Linux-VServer had a similar result for read and re-read operations. Otherwise for write operations, where the VServer slightly exceeds the native performance, and LXC that reaches a near-native performance. The same behavior obtained for write operations regarding to Linux-VServer were similar as describe in [25]. Comparing these results with OpenVZ, we observed a gain of performance. We believe that it is due the I/O scheduler used by the different systems. While LXC and Linux-VServer use the "deadline" linux scheduler [26], OpenVZ uses CFQ scheduler in order to provide the container disk priority functionality. The "deadline" scheduler imposes a deadline on all I/O operations to ensure that no request gets starved, and aggressively reorders requests to ensure improvement in I/O performance. The worst result was observed in Xen for all I/O operations due to the para-virtualized drivers. These drivers are not able to achieve a high performance yet.

D. Network Performance

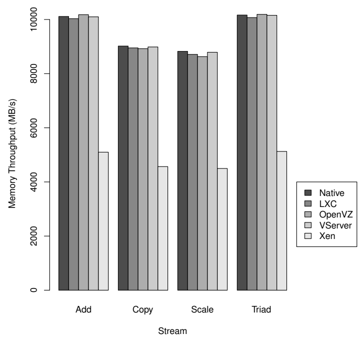
The network performance was evaluated with the NetPIPE (Network Protocol Independent Performance Evaluator) [27] benchmark. NetPIPE is a tool for measurement of network performance under a variety of conditions. It performs simple tests such as ping-pong, sending messages of increasing size between two processes, either through a network or a multiprocessor architecture. The message sizes are chosen and sent at regular intervals to simulate disturbances and provide a complete test of the communication system. Each data point involves many ping-pong tests to provide accurate time measurements, allowing the calculation of latencies.

Figure 3 shows the comparison of the network bandwidth in each virtualization system. The Linux-VServer obtained similar behavior to the native implementation, followed by LXC and OpenVZ. The worst result was observed in Xen. Its average bandwidth was 41% smaller the native, with a maximum degradation of 63% for small packets. Likewise, the network latency presented in Figure 4 shows that Linux-VServer has near native latency. The LXC again has a great score, with a very small difference when compared to Linux-VServer and native systems, followed by OpenVZ. The worst latency was observed in Xen.

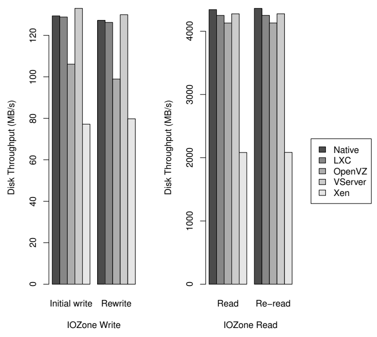
These results can be explained due to different implementations of the network isolation of the virtualization systems. While Linux-VServer does not implement virtualized network devices, both OpenVZ and LXC implement network namespace that provides an entire network subsystem. We did not configure a real network adapter in OpenVZ system, as described in Section II, because it would reduce the scalability due the limited number of network adapter that normally exist in host machine. The Xen network performance degradation is caused by the extra complexity of transmit and receive packets. The long data transfer path



(a) Computing performance using Linpack for matrices of order 3000.



(b) Memory throughput using STREAM.



(c) Disk throughput using IOZone.

Figure 2. Performance evaluation for different micro-benchmarks

between the guests and the hypervisor adversely impact the throughput [28].

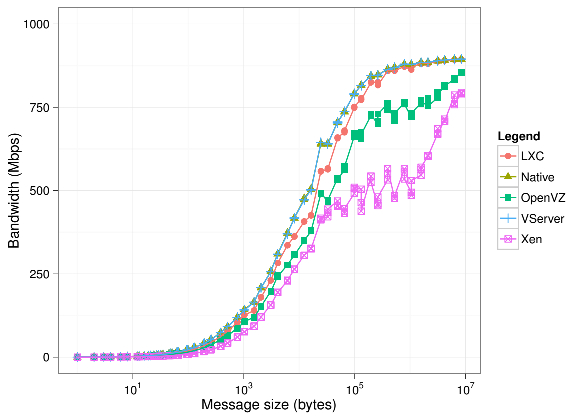


Figure 3. Network bandwidth using NetPIPE.

E. Performance Overhead in HPC Applications

This section presents an analysis of the performance overhead in container-based systems for HPC applications. For that, we conducted experiments using the NPB benchmark suite [14]. NPB is derived from computational fluid dynamics (CFD) applications and consist of five kernels (IS, EP, CG, MG, FT) and three pseudo-applications (BT, SP, LU).

The first experiment aimed to evaluate the performance overhead of the virtualization system in a singlenode environment. Figure 5 shows the results for each NPB benchmark using its OpenMP implementation. In all cases, the container-based systems obtained execution times very close to the native system. However, among the container-based systems evaluated, OpenVZ had the worst performance,

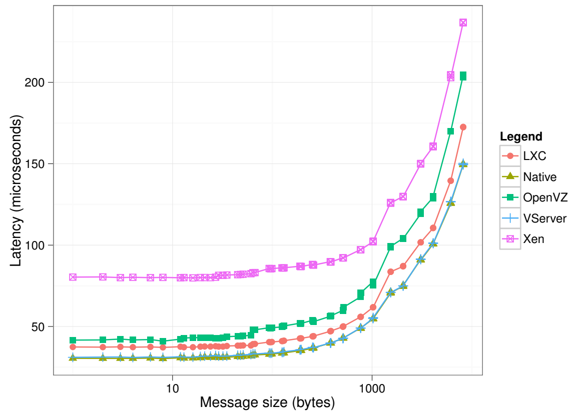


Figure 4. Network latency using NetPIPE.

specially for the benchmarks BT and LU. This is due to these benchmarks make intensive use of memory and cause many cache misses and some implementations of container-based systems, such as OpenVZ, have limited cache memory blocks. Also, Xen shows the problem of double-cache, as described in Section III-B. The results of Xen show that it was only able to achieve near native performance for CPU-intensive benchmarks, such as EP, FT and IS.

When evaluating a multinode, we can see in Figure 6 differences become more evident, since several of the benchmarks used are tests that include network, such as CG, FT and MG. The Xen obtained the worst performance among all the virtualization systems probably due to network driver virtualized, as observed in network performance tests.

In singlenode, the differences are not so expressive, with some differences only in benchmarks that used intensively the cache. When evaluating a multinode environment, the influence of the network can be noted as the primary metric

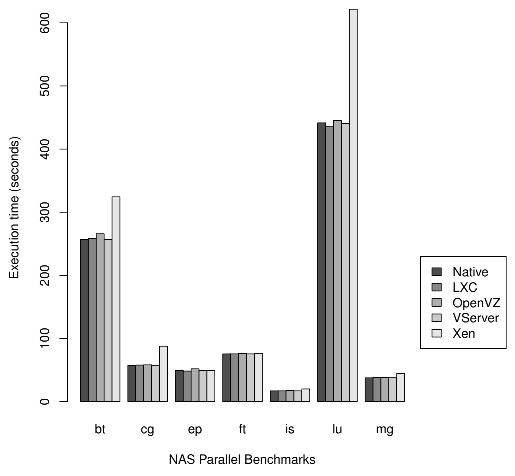


Figure 5. Single node comparison of NPB performance overhead.

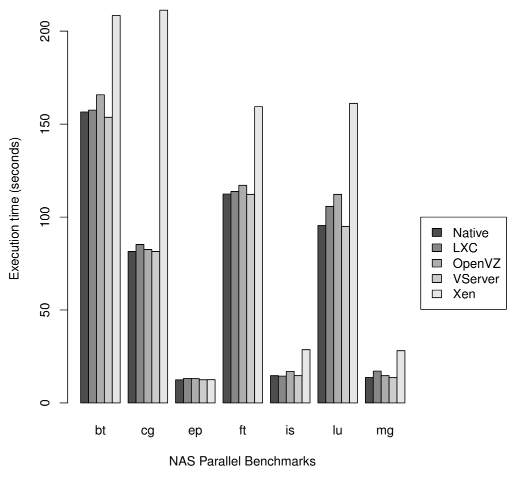


Figure 6. Multi node comparison of NPB performance overhead.

to be discussed, since the network has a direct impact in isolation.

F. Isolation

To analyze the best results of isolation between the different systems, we use the Isolation Benchmark Suite (IBS) [29] that demonstrates how much a virtualization system can limit the impact of a guest with other guest running on a single host machine. This set of benchmarks includes six different stress tests: CPU intensive test, memory intensive test, a fork bomb, disk intensive test and two network intensive tests (send and receive). To perform the isolation tests, we construct environments with two guests on the same host machine, and the resources of the host machine were divided by two and assigned on each guest. Hence. we will present our detailed analysis of the isolation performance for Linux-VServer, OpenVZ, LXC and Xen. The resource limits of each environment were set based on techniques available for each virtualization system. In both Linux-VServer and LXC, the resource limits have been set up through the cgroups, which restrict the usage of the defined CPU's and the memory consumption for each guest. In OpenVZ we used the vzsplit tool that divides the total amount of available resources between guests. Lastly, in Xen environment we create two guest virtual machines on the host machine to reflect the configuration of the container-based environments.

First of all, we collected the execution time of a given baseline application upon one of the guest, we chose the LU pseudo application of NPB benchmark. After completing the baseline measurements, we ran the baseline application on all guests an then in addition introduce a stress test in one of the guest. We collected the application execution time of the well-behaved guest. The IBS isolation metric was obtained comparing the application execution time that was ran into one guest against the application execution time obtained from the well-behaved guest while the stress test was performed. Finally, we quantify the performance degradation of the well-behaved guest.

As shown in Table I, all systems had no impact in CPU intensive test. It demonstrates that the CPU affinity configured by cgroup, and the VCPU Affinity technique used in OpenVZ environment are working well. However, all others resources when stressed had some impact in well-behaved guests. As described in Section II, all virtual instances in container-based systems shares a single operating system kernel. Hence, we supposed that while the kernel needs to handle instruction calls from the stressed guest, it is unable to handle instruction calls from the well-behaved guest. This behavior could has influenced the performance for all virtualization systems while the memory, disk and network tests were performed. The fork bomb test is a classic test that loops creating new child processes until there are no resources available. The fork bomb test demonstrates that exist security failures on LXC system, due to the impossibility to limit the number of processes by cgroups. Both Linux-VServer and OpenVZ use their own implementations in order to limit the number of processes. As result, the well-bahaved guests are not impacted by the stress test. Our tests also demonstrate that the Xen has the better isolation, due to non shared operating system.

IV. Related Work

Many papers have studied the performance overheads of virtualization technologies, a few of them focusing on HPC environments. Walters et al. [7] evaluated the performance of VMware Server, Xen, and OpenVZ for HPC, using the NPB benchmark (using both OpenMP and MPI) and micro-benchmarks for network and disk. In their experiments, both Xen and OpenVZ achieved near-native performance for the CPU intensive benchmarks, but OpenVZ outperformed Xen

Table I
PERFORMANCE ISOLATION FOR LU APPLICATION. THE RESULTS REPRESENT HOW MUCH THE APPLICATION PERFORMANCE IS IMPACTED BY DIFFERENT STRESS TESTS IN ANOTHER VM/CONTAINER. DNR MEANS THAT APPLICATION WAS NOT ABLE TO RUN.

	LXC	OpenVZ	VServer	Xen
CPU Stress	0	0	0	0
Memory	88.2%	89.3%	20.6%	0.9%
Disk Stress	9%	39%	48.8%	0
Fork Bomb	DNR	0	0	0
Network Receiver	2.2%	4.5%	13.6%	0.9%
Network Sender	10.3%	35.4%	8.2%	0.3%

for the network intensive ones. VMware Server had the worst performance in all cases. Their experiments were only focused on performance.

Regola and Ducom [6] evaluated KVM, Xen and OpenVZ for HPC. They also used the NPB benchmark (using both OpenMP and MPI) and micro-benchmarks for network and disk. Their experiments included instances of the Amazon's EC2 "Cluster Compute Node" service, which is supposed to use Xen as underlying hypervisor and a 10 Gbit/s Ethernet for high throughput communication. Again, all virtualization systems obtained near-native performance for CPU intensive benchmarks. OpenVZ had the best performance for I/O intensive benchmarks. Their experiments using EC2 instances showed that the Amazon's service was not able to delivery the full network capacity, resulting in a poor performance in the network benchmark. No experiments to evaluate the isolation were performed for any of the systems.

Soltesz et al. [25] presented the design and implementation of Linux-VServer and compared it with Xen. They did not use any HPC workloads in this evaluation. Instead, their experiments used a benchmark for database servers and micro-benchmarks for CPU, disk and network. Their results showed that Linux-VServer provides comparable support for isolation and superior performance than Xen. The network performance in Linux-VServer was significantly better than in Xen.

It seems to be a consensus that today's hypervisor-based virtualization systems perform well for CPU intensive applications, but exhibit a high overhead when handling I/O intensive applications, especially the network intensive ones [30], [31]. However, there are some alternatives to avoid the high overhead in network operations in virtualized systems. For example, Liu et al. [30] presented the idea of VMM-bypass, which extends the original idea of OS-bypass to VM environments. Basically, it allows time-critical I/O operations to be carried out directly in guest VMs without any involvement of the VMM. When applied to networking, VMM-bypass allows, for example, the direct access to high performance network devices, such as InfiniBand, resulting in a near-native network performance [30]. We conducted our experiments without any special configuration to bypass VMM for I/O operations in Xen.

As discussed early in this paper, isolation is an important concern in virtualization for HPC environment, specially for the case of sharing multicore machines between multiples users. However, we found little work evaluating it. Deshane et al. [32] proposed the Isolation Benchmark Suite (IBS) that quantifies the degree to which a virtualization system limits the impact of a misbehaving VM on other well-behaving VM running on the same physical machine. They also evaluated the performance isolation for VMware Workstation, Xen and OpenVZ. We reproduced their experiments in this paper, but this time including Linux-VServer and LXC, and obtained results similar to theirs, i.e., hypervisor-based virtualization provides better isolation than container-based. Matthews et al. [15] also used IBS to compare the isolation performance of Xen and KVM. They reported that KVM had an unexpectedly poor performance for disk and network isolation tests.

We believe our work is complementary to the works presented in this section. We evaluated the performance and isolation of container-based virtualization for HPC environments. Our experiments cover all the important parts of an HPC platform (CPU, memory, disk and network) and the main software technologies for HPC (OpenMP and MPI). Moreover, this is the first work to perform an in-depth performance evaluation of LXC, including isolation measurements and tests with typical HPC workloads. Also, to the best of our knowledge, this is the first work to perform a comparison of the three current container-based implementation: LXC, OpenVZ and Linux-VServer.

V. CONCLUSION AND FUTURE WORK

We presented container-based virtualization as an lightweight alternative to hypervisors in HPC context. As we have shown, there are useful usage cases in HPC where both performance and isolation are need. In that way, we conducted experiments to evaluate the current Linux container-based virtualization implementations and compare them to Xen, a commonly used hypervisor-based implementation.

HPC clusters are typically shared among many users or institutes, which may have different requirements in terms of software packages and configurations. As presented, such platforms might benefit from using virtualization technologies to provide better resource sharing and custom environments. However, HPC will only be able to take advantage of virtualization systems if the fundamental performance overhead (such as CPU, memory, disk and network) is reduced. In that sense, we found that all container-based systems have a near-native performance of CPU, memory, disk and network. The main differences between them lies in the resource management implementation, resulting in poor isolation and security. While LXC controls its resources only by cgroups, both Linux-VServer and OpenVZ implement their own capabilities introducing even more resource limits, such as the number of processes, which we have found to be an important contribution to give more security to the whole system. We suppose this capability will be introduced in cgroups in a near future.

Careful examination of the isolation results reveals that all container-based systems are not mature yet. The only resource that could be successfully isolated was CPU. All three systems showed poor performance isolation for memory, disk and network. However, for HPC environments, which normally does not require the shared allocation of a cluster partition to multiple users, this type of virtualization can be very attractive due to the minimum performance overhead.

Since the HPC applications were tested, thus far, LXC demonstrates to be the most suitable of the container-based systems for HPC. Despite LXC does not show the best performance of NPB in multinode evaluation, its performance issues are offset by the easy of and management. However, some usual virtualization techniques that are useful in HPC environments, such as live migration, checkpoint and resume, still need to be implemented by the kernel developer team.

As future work, we plan to study the performance and isolation of container-based systems for other kind of workloads, including I/O bound applications. For example, data-intensive applications such as those based on the MapReduce model.

References

[1] "Xen," 2012. [Online]. Available: http://www.xen.org

[2] "VMware," 2012. [Online]. Available: http://www.vmware.com

[3] "KVM," 2012. [Online]. Available: http://www.linux-kvm.org

[4] "Virtualization Technology," 2012. [Online]. Available: http://ark.intel.com/Products/VirtualizationTechnology

[5] "AMD Virtualization," 2012. [Online]. Available: http://www.amd.com/br/products/technologies/virtualization/

[6] N. Regola and J.-C. Ducom, "Recommendations for virtualization technologies in high performance computing," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 30 2010-dec. 3 2010, pp. 409 –416.

[7] J. P. Walters, V. Chaudhary, M. Cha, S. G. Jr., and S. Gallo, "A comparison of virtualization technologies for hpc," *Advanced Information Networking and Applications, International Conference on*, vol. 0, pp. 861–868, 2008.

[8] "Linux VServer," 2012. [Online]. Available: http://linux-vserver.org

[9] "OpenVZ," 2012. [Online]. Available: http://www.openvz.org

[10] "Linux Containers," 2012. [Online]. Available: http://lxc.sourceforge.net

[11] "TORQUE Resource Manager," 2011. [Online]. Available: http://www.clusterresources.com/products/torque-resource-manager.php

[12] 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," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 22–22.

[13] K. Chen, J. Xin, and W. Zheng, "Virtualcluster: Customizing the cluster environment through virtual machines," in *Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on*, vol. 2, dec. 2008, pp. 411 –416.

[14] "Nas parallel benchmarks," 2012. [Online]. Available: http://www.nas.nasa.gov/publications/npb.html

[15] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," in *Proceedings of the 2007 workshop on Experimental computer science*, ser. ExpCS '07. New York, NY, USA: ACM, 2007.

[16] E. W. Biederman, "Multiple Instances of the Global Linux Namespaces," in *Proceedings of the Linux*, 2006.

[17] "Control groups definition, implementation details, examples and api," 2012. [Online]. Available: http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[18] P. P. Tang and T.-Y. Tai, "Network traffic characterization using token bucket model," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, mar 1999, pp. 51 –62 vol.1.

[19] "Credit-based cpu scheduler," 2012. [Online]. Available: http://wiki.xensource.com/xenwiki/CreditScheduler

[20] "Inside the linux 2.6 completely fair queueing," 2012. [Online]. Available: http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp

[21] N. Regola and J.-C. Ducom, "Recommendations for virtualization technologies in high performance computing," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 30 2010-dec. 3 2010, pp. 409 –416.

[22] "LINPACK Benchmark," 2012. [Online]. Available: http://www.netlib.org/benchmark

[23] "STREAM Benchmark," 2012. [Online]. Available: http://www.cs.virginia.edu/stream/

[24] "IOzone Filesystem Benchmark," 2012. [Online]. Available: http://www.iozone.org

[25] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 275–287, Mar. 2007.

[26] "Choosing an i/o scheduler for red hat enterprise linux 4 and the 2.6 kernel," 2012. [Online]. Available: http://www.redhat.com/magazine/008jun05/features/schedulers

[27] "Network Protocol Independent Performance Evaluator," 2012. [Online]. Available: http://www.scl.ameslab.gov/netpipe

[28] 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," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 164–177.

[29] "Isolation Benchmark Suite," 2012. [Online]. Available: http://web2.clarkson.edu/class/cs644/isolation

[30] J. Liu, W. Huang, B. Abali, and D. K. Panda, "High performance VMM-bypass I/O in virtual machines," in *Proceedings of the annual . . .*, 2006.

[31] P. Apparao, S. Makineni, and D. Newell, "Characterization of network processing overheads in xen," in *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, ser. VTDC '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 2–.

[32] T. Deshane, Z. Shepherd, and J. Matthews, "Quantitative comparison of Xen and KVM," *Xen Summit*, 2008.