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

Miguel Gomes Xavier, Marcelo Veiga Neves, Fabio Diniz 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@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 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], [8]. 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 [9], OpenVZ [10] and Linux Containers (LXC) [11]) offer a lightweight virtualization layer, which promises a near native performance.

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.

We conducted experiments using the NAS Parallel Benchmarks (NPB) [12], which is a well-known benchmark in HPC, to evaluate the performance overhead and the Isolation Benchmark Suite (IBS) [13] 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.

This paper is organized as follows: Section II provides an overview of virtualization techniques and a comparison between traditional hypervisor-based virtualization and container-based virtualization; Section III presents the experiments performed in order to evaluate both performance overhead and isolation; Section IV presents the related work. The conclusion and future work are presented in Section V.

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 a 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.

Since the container-based virtualization works at the operating system level, all virtual instances share a single

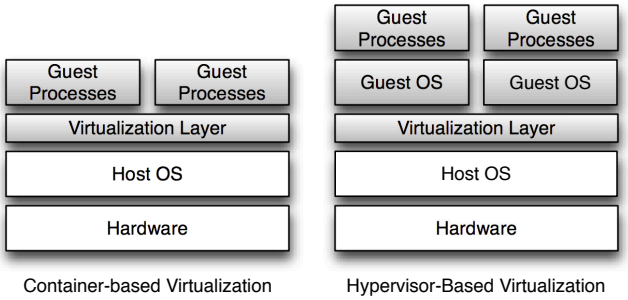


Figure 1. Comparison of Container-based and Hypervisor-based virtualization.

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 [10].

The isolation in container-based virtualization is normally done by kernel namespaces [14]. It is a feature of the Linux kernel that allow 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 [14].

On the other hand, the resources management in container-based virtualization systems is normally done by Control Groups (cgroup) [15], which restricts a resource usage for 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 the container-based virtualization systems studied in this work, which are Linux-VServer, OpenVZ and LXC.

A. Linux-VServer

Linux-VServer is 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. The 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 benefits of this approach is the 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 [9].

The 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 another container, 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 [9].

To perform the CPU isolation, Linux-VServer uses the standard Linux scheduler, overlapped by the Token Bucket Filter (TBF) [16] 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 [9]. The token bucket scheme is very similar to Xen Credit Scheduler [17].

The 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 [9].

B. OpenVZ

OpenVZ offers some similar functionality to Linux-VServer. However, builds on 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. This includes network devices, routing tables, firewall rules and so on. The system 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 [10].

The system introduces four resource management components named as User Beancounters (UBS), far 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 as VCPU Affinity, which tells the kernel the maximum number of CPU's 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 [10]. 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 [18]. 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 [10].

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, the 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, the system uses the network namespaces. Two configuration are offer 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. In network perspective, cgroups defines the configuration of network namespaces [11]. The system uses multiple controllers over the standard linux CPU scheduler. The process control is 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. In this system, the containers are controled by the lxc-tool [11].

III. Experiments

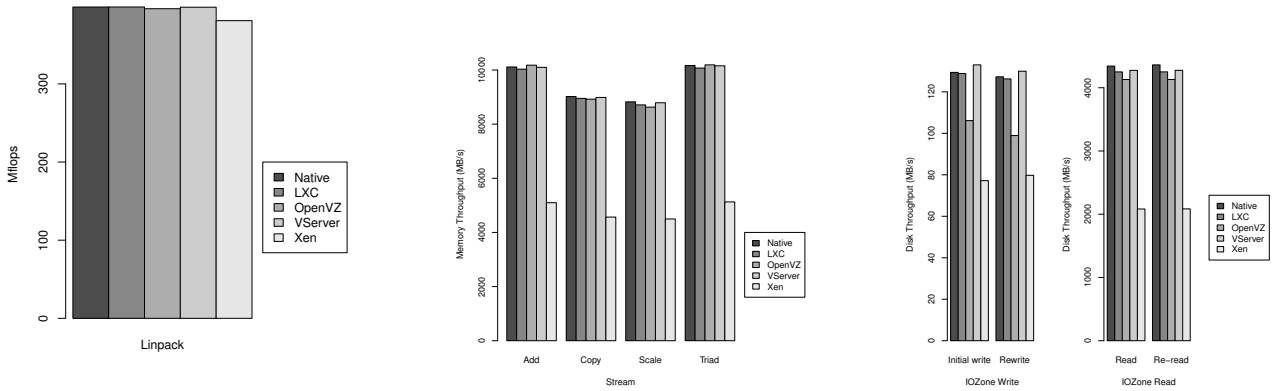
This section studies the performance and isolation of container-based and hypervisor-based virtualization. We performed several experiments with the current linux container-based virtualization implementations: Linux VServer, OpenVZ and LXC. We also chose Xen as the representative of hypervisor-based virtualization, because it is considered one of the most mature and efficient implementations of this kind of virtualization [6].

Our experimental setup consists of four identical Dell PowerEdge R610 with two 2.27GHz Intel Xeon E5520 processors (with 8 cores each), 8M of L2 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) 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 and losses of performance that would influence the results of experiments. 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 developer team [10], 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. Hence, we just need to install the LXC tool package (0.6.5-1) and ensured that all requirements showed 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 [19]. 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%.



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

(b) Memory throughput using STREAM.

(c) Disk throughput using IOZone.

Figure 2. Analyze of performance for different benchmarks

B. Memory Performance

The memory performance on a single node was evaluated with STREAM [20], 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, the Xen shows the problem of double-cache, the same blocks that are used by the host and the virtual machine.

C. Disk Performance

The disk performance was evaluated with the IOzone benchmark [21]. 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 [22]. Comparing these results with OpenVZ, we observed a gain of performance and 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 [23], 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 the 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) [24] 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 to provide a complete test of the communication system. Each data point involves many ping-pong tests to provide accurate time, 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 between the guests and the hypervisor adversely impact the throughput [25].

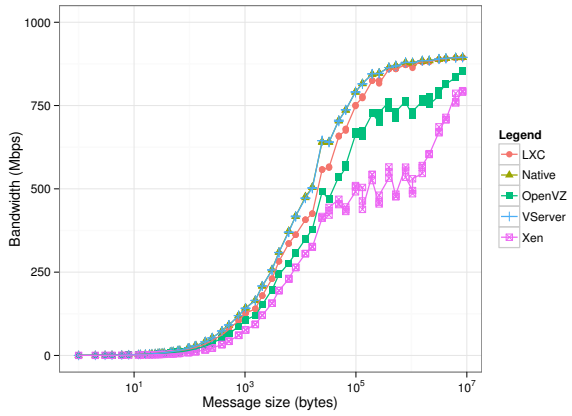


Figure 3. Network bandwidth using NetPIPE.

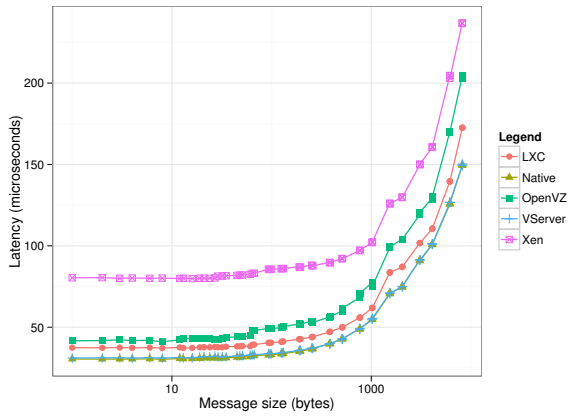


Figure 4. Network latency using NetPIPE.

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 [12]. 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, specially for the benchmaks BT and LU. This is due to these benchmarks make intensive use of memory and cause many cache misses and some implementations of containter-based systems, such as OpenVZ, have limited cache memory blocks. Also, the Xen shows the problem of double-cache, the same blocks that are used by the host and the virtual machine. 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.

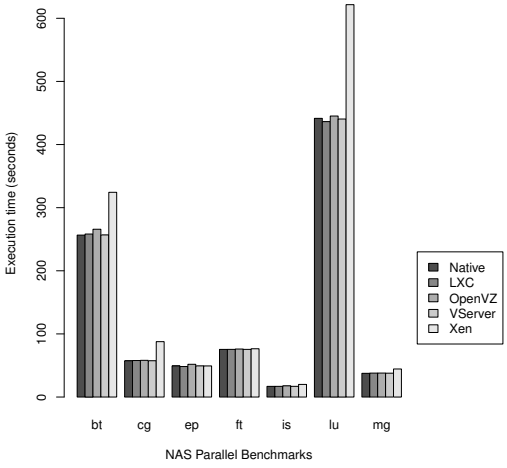


Figure 5. Single node comparison of NPB performance overhead.

Since 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 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) [26] 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

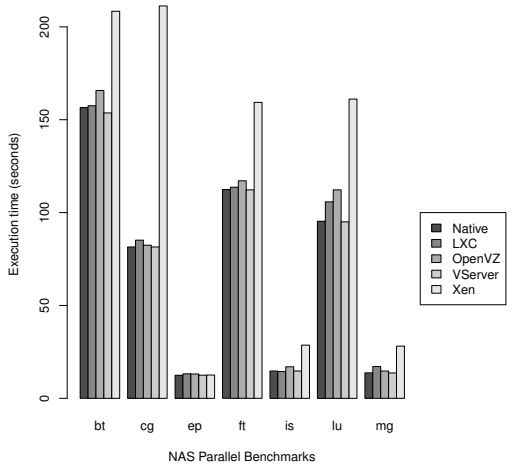


Figure 6. Multi node comparison of NPB performance overhead.

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 avaliable 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 meansurements, 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 calls instructions from the stressed guest, it is unable to handle call instructions from the well-behaved guest. This behavior could has influenced the performance for all vitualization systems while the memory, disk and the network tests were performed. The fork bomb test is a classic fork test that loops creating new child processes until there are no resources avaliable. The fork bomb test demonstrates that exist security fails 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.

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%

IV. RELATED WORK

Some papers have explored the overhead of virtualization as presented by Apparao et al. [27] and Menon et al. [28] The results showed that the overhead of the virtualization layer on the CPU and I/O can not be eliminated, no matter the virtualization model used (operating system virtualization, para-virtualization and full virtualization). There are several studies showing the impact of virtualization on the CPU, proposing some improvements. Cherkasova et al. [29] compares three Xen schedulers on different workloads and shows the overhead of virtualization on each one.

The major limitation of virtualization technology is its ability to handle I/O intensive applications. Ranadive et al. [30] proposed a bypass to allow direct access to device resources as Infiniband devices. We can see in Dong et al. [31] the development of new techniques for improving I/O in a virtualized environment based on input/output memory management unit. Thus, presents a significant gain by using this technique, getting very close to the native system in some cases. In the same way, Himanshu Raj and Karsten Schwan [32] developed a new approach for virtualizing I/O called self-virtualized devices. Hence, proposal tries to improve the performance I/O virtual machines interact with the virtualized devices. The focus of work was the network device in an attempt to reduce its overhead by domains.

In Walters [33], Vallee [34] and Mergen [35], the specific focus was the virtualization of HPC environments, so that the virtualization is shown as an alternative for intelligent use of available resources, because there are many advantages. This papers show the important characteristics of virtualized environments and some advantages when used in HPC environments. Our work shows an alternative to hypervirosr-based virtualizastion environments, presenting evaluating overhead using different metrics on some container-based virtualization. We have some previous results showing that the container-based virtualization have less overhead than hypervisor-based environments.

Besides improving the overhead, another important point of virtualization consists in isolation between virtual machines. Matthews et al. [36] reported the isolation between virtual machines and its impacts on environmental performance. Isolation performance was also evaluated in our work, and we can see that although the container-based virtualization provides less overhead than hypervisor-based environments, there are still some problems of isolation.

However, container-based virtualization has better CPU and I/O performance, close to the native model. Soltesz et al. [37] presents containers as an alternative to using virtualization. Shown a comparison between Linux-VServer and Xen virtual machine monitor, allowing analysis of certain metrics such as isolation. Tests were performed on PlanetLab, an environment that allows experiments on a global scale, using benchmarks. The results of these tests show that container comes from better isolation when compared to virtual machines, managing resource limitations in a better way. Our study presented an analysis of the different container-based virtualization using benchmarks, and by the results we could verify what type of systems may cause less overhead.

Regola and Ducom [38] show an evaluation of three known virtualization systems: KVM, OpenVZ and Xen. They used workloads of HPC were using OpenMP and MPI, with specific focus on the overall performance of the environment, without concern for isolation and scalability.

Our proposal is complementary to the works presented in this section because we analyze the container-based virtualization in HPC environment. Moreover, the performance evaluated in our work covers several metrics such as: CPU, file system, disk and network. This study provides analysis of performance and isolation. Therefore, this paper presents some important contributions, it is the first work that presents performance evaluations of the LXC system. 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 isolation and performance are needed. 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. However, HPC will only be able to take advantage of virtualization systems if the fundamental performance overhead (such as CPU, memory, I/O and network) is reduced. Hence, we found that all container-based systems have a near native performance of CPU, memory, I/O and network. The main differences between them lies in the resource management implementation, resulting in poor isolation and security. While LXC system 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 is the main 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 isolation results reveals that all container-based systems are not mature yet. However, for HPC environments, which does not require resource sharing, 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 mature of container-based systems for HPC. Despite LXC does not show the best performance of NPB in multinode evaluation, the low overhead is offset by the use facility 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 of other kind of workloads, such as data-intensive applications based on MapReduce, due to its I/O bound characteristics.

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] V. Chaudhary, M. Cha, J. Walters, S. Guercio, and S. Gallo, "A comparison of virtualization technologies for hpc," in *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, march 2008, pp. 861 –868.

[8] G. Vallee, T. Naughton, C. Engelmann, H. Ong, and S. Scott, "System-level virtualization for high performance computing," in *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, feb. 2008, pp. 636 –643.

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

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

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

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

[13] 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*. ACM, jun 2007.

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

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

[16] 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.

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

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

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

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

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

[22] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07. New York, NY, USA: ACM, 2007, pp. 275–287.

[23] "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

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

[25] 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.

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

[27] 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–.

[28] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the xen virtual machine environment," in *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, ser. VEE '05. New York, NY, USA: ACM, 2005, pp. 13–23.

[29] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three cpu schedulers in xen," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 2, pp. 42–51, Sep. 2007.

[30] A. Ranadive, A. Gavrilovska, and K. Schwan, "Ibmon: monitoring vmm-bypass capable infiniband devices using memory introspection," in *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*, ser. HPCVirt '09. New York, NY, USA: ACM, 2009, pp. 25–32.

[31] Y. Dong, J. Dai, Z. Huang, H. Guan, K. Tian, and Y. Jiang, "Towards high-quality i/o virtualization," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, ser. SYSTOR '09. New York, NY, USA: ACM, 2009, pp. 12:1–12:8.

[32] H. Raj and K. Schwan, "High performance and scalable i/o virtualization via self-virtualized devices," in *Proceedings of the 16th international symposium on High performance distributed computing*, ser. HPDC '07. New York, NY, USA: ACM, 2007, pp. 179–188.

[33] 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.

[34] G. Vallee, T. Naughton, C. Engelmann, H. Ong, and S. L. Scott, "System-level virtualization for high performance computing," in *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, ser. PDP '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 636–643.

[35] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for high-performance computing," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 2, pp. 8–11, Apr. 2006.

[36] 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.

[37] 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.

[38] 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.