2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

A Performance Comparison of Container-based Virtualization Systems for MapReduce Clusters

Miguel G. Xavier, Marcelo V. Neves, Cesar A. F. De Rose
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, Brazil
miguel.xavier@acad.pucrs.br

***Abstract*—Virtualization as a platform for resource-intensive applications, such as MapReduce (MR), has been the subject of many studies in the last years, as it has brought benefits such as better manageability, overall resource utilization, security and scalability. Nevertheless, because of the performance overheads, virtualization has traditionally been avoided in computing environments where performance is a critical factor. In this context, container-based virtualization can be considered a lightweight alternative to the traditional hypervisor-based virtualization systems. In fact, there is a trend towards using containers in MR clusters in order to provide resource sharing and performance isolation (e.g., Mesos and YARN). However, there are still no studies evaluating the performance overhead of the current container-based systems and their ability to provide performance isolation when running MR applications. In this work, we conducted experiments to effectively compare and contrast the current container-based systems (Linux VServer, OpenVZ and Linux Containers (LXC)) in terms of performance and manageability when running on MR clusters. Our results showed that although all container-based systems reach a near-native performance for MapReduce workloads, LXC is the one that offers the best relationship between performance and management capabilities (specially regarding to performance isolation).**

***Keywords*-Container-based virtualization; High performance computing; MapReduce**

I. Introduction

MapReduce (MR) has become a very popular programming model for large-scale data analysis. Such popularity is associated with the growing volume of data we face today and the ability of MR to process such a large data set efficiently. The more amount of data, the more the need for computational resources. In such a way, data processing systems, such as Hadoop [1], have been proposed to simplify the concepts involved in large-scale distributed computing and offer ease of programming, scalability and fault tolerance. By implementing MR, Hadoop has overcome barriers, solving heavy problems containing large volume of data and achieving results in units of times earlier impossible. Hadoop is currently the most popular and widely deployed of the open-source MR implementations. It was primarily supported by Yahoo! and later also used by many Internet companies, including Twitter, Facebook [2] and Amazon [3].

Such growing of data can be clearly observed in the context of cloud computing. Users are increasingly using cloud services to store large amounts of data and take advantage of benefits such as scalability, cost efficiency, manageability, fault tolerance and greater computing feasibility. To fulfil these issues, Hadoop on cloud computing has allowed providers to provide on-demand computing capacity, which were limited just on the instance on-demand model. For example, Amazon has recently launched the EMR [3] instance, which allows businesses, researchers and developers to easily and cost-effectively process a vast amount of data through a framework hosted in a web-scale infrastructure [3]. Since cloud computing platforms are in most cases supported by hypervisor-based virtualization systems, such as KVM [4] and Xen [5], Hadoop applications on such systems are normally unable to reach the same performance obtained from traditional non-virtualized platforms because of their inherent performance overhead [6][7]. That is why, for example, hypervisor-based virtualization has traditionally been avoided in traditional High Performance Computing (HPC) clusters [7].

On the other hand, container-based virtualization systems (e.g., Linux-VServer [8], OpenVZ [9] and Linux Containers (LXC) [10]) prove to be an alternative to hypervisors-based systems, providing better manageability with a near-native performance. In fact, there is a trend toward using this type of virtualization in MR cluster, since it has brought new opportunities for sharing environments in which users' requirements differ between one another. For example, users have developed a diverse array of computing frameworks, such as MPI [11] and Hadoop, in order to simplify programming and to better fit parallel applications. To prevent the mismatch between such clusters and existing frameworks, a fine-grained manager system is needed [12]. In this context, Mesos [13] has emerged as a platform for sharing commodity clusters between multiple diverse cluster computing frameworks while incurring low overheads due of the container-based virtualization system adopted. Moreover, the use of container-based virtualization to share a cluster among different applications will become a built-in feature in the next generation of Hadoop [14], called MapReduce 2.0 (MRv2) or YARN (Yet Another Resource Negotiator). However, there are still no studies evaluating the performance overhead of such container-based systems and their ability to provide performance isolation for shared resources

1066-6192/14 $31.00 © 2014 IEEE
DOI 10.1109/PDP.2014.78

when running MapReduce applications.

On our previous work [7], we have demonstrated through a variety of experiments that container-based systems in HPC clusters offer several benefits while with a low performance overhead in MPI and OpenMP applications. Herein we conducted experiments using different MR benchmarks to stress Hadoop clusters over distinct situations on the recent container-based virtualization systems: Linux-VServer, OpenVZ and LXC. We also examined the performance isolation of LXC, which is the most recent container-based system and is currently used by Mesos and YARN.

This paper is organized as follows: Section II provides an overview of the MR model and its implementations; Section III presents a summary of the recent container-based virtualization systems; Section IV describes the experiments and discusses the results. Section VII discusses the related work. The conclusion and future work are presented in Section VIII.

II. MapReduce

MapReduce is a programming model and a framework that supports this model. It simplifies the concepts involved in large-scale distributed computing and offers ease of programming, scalability and fault tolerance. MR implementations are typically coupled with a distributed file system (DFS), such as GFS [15] or HDFS [16]. The DFS is responsible for the data distribution in a MR cluster, which consists of initially dividing the input data into blocks and storing multiple replicas of each block on the cluster nodes' local disks. Although there are currently several implementations of MapReduce (e.g., Hadoop [1], Twister [17], Spark [18]), this work will focus on Hadoop because it is one of the most popular open-source MapReduce implementations. Moreover, there are a variety of software that run on top of the Hadoop stack, which create an entire ecosystem of big data processing tools. The Hadoop software can be roughly divided into two main components: Hadoop MapReduce and Hadoop Distributed File System (HDFS).

A. Hadoop MapReduce

Hadoop MapReduce is a distributed programming framework and an execution environment for MapReduce programs. The execution environment also includes a job scheduling system that coordinates the execution of multiple MapReduce programs, which are submitted as batch jobs. A MR job consists of multiple map and reduce tasks that are scheduled to run in the Hadoop cluster's nodes. Multiple jobs can run simultaneously in the same cluster. There are two types of nodes that control the job execution process: a JobTracker and a number of TaskTrackers [19]. Users submit MR jobs to the JobTracker, which is responsible for coordinating the execution of all the jobs in the system. The JobTracker schedules tasks to run on TaskTrackers, which have a fixed number of slots to run the map and reduce tasks. TaskTrackers run tasks and report the execution progress back to the JobTracker, which keeps a record of the overall progress of each job. The JobTracker always tries to assign tasks to the TaskTrackers that are the closest to the input data.

B. Hadoop Distributed File System

Hadoop Distributed File System (HDFS) [16] is a distributed file system designed to store very large files and to provide high-throughput for streaming data access patterns. All application data in Hadoop is stored as HDFS files, which are composed of data blocks of a fixed size (64 MB each, by default) distributed across multiple nodes. There are two types of nodes in a HDFS cluster: a NameNode and a number of DataNodes. The NameNode maintains the file system metadata, which includes information about the files and directories tree as well as where each data block is physically stored. DataNodes store the data blocks themselves. Every time a client needs to read a file from HDFS, it first contacts the NameNode to determine the DataNodes where all the blocks for that file are located. Then, the client starts reading the data blocks directly from the DataNodes.

Each data block is independently replicated (typically 3 replicas per block) and stored within multiple DataNodes. The replicas' placement follows a well-defined rack-aware algorithm that uses the information of where each DataNode is located in the network topology to decide where data replicas should be placed in the cluster. Basically, for every block of data, the default placement strategy is to place two replicas on two different nodes on a same rack and the last on a node on a different rack. Replication is used not only for providing fault tolerance, but also to increase the opportunity for scheduling tasks to where the data is, by spreading replicas out on the cluster.

III. Container-based Virtualization

Container-based Virtualization is a virtualization architecture which promise to be a lightweight alternative to hypervisor-based architecture. While hypervisors work at the hardware abstraction level, container-based virtualization is characterized as multiple isolated user-spaces running at the operational system level, providing abstractions directly to the guest processes. For this reason, it is also commonly knows as Operating System Level virtualization. Nowadays, the most representative implementations of container-based virtualizations are: Linux-Vserver, OpenVZ and LXC. All of them are Linux implementations and have some similarity when aspects such as security, isolation and performance are taken into account. The main difference between them lies in the way resources are managed, such as the manner of how the resources are limited between multiple containers on a single machine and how the resource isolation is accomplished.

Because container-based virtualization works at the operating system level, all instances (containers) share the same operational system kernel. That is why container-based virtualization is supposed to have a weaker isolation when

compared to hypervisor-based virtualization. From the user' point of view, the containers execute exactly like a stand-alone OS [9].

In order to guarantee the resource isolation between the host system and the containers running on, such a system implements kernel namespaces, which has been incorporated as features in the Linux mainline kernel by steps since 2.6.19 [20]. Namespace allows different processes to have a different view of the system. Once containers should not be able to interact with things outside, many global resources are wrapped in a layer of namespace that provides the illusion to the user that container is its own subsystem [20]. As examples of the most notables namespaces supported by the recent Linux kernel versions, consider: Filesystem namespace, Process IDs (PID) namespace, Inter-process communication (IPC) namespace and Network namespace.

In container-based systems, it is also possible to restrict the resource usage by each container, which provides a fair distribution of the overall available resources from a single machine. This resource management is normally accomplished by Control Groups (cgroup) [21], which restricts the resource usage per group of processes. For example, using cgroups it is possible to limit/prioritize CPU, memory and I/O usage for different containers. In some cases, some container-based systems use their own implementations to perform resource management due to its incompatibility with cgroups.

The remainder of the paper presents at in more details the systems explored in this work: Linux-VServer, OpenVZ and LXC. The substantial aspects regarding the resource management and techniques for limiting resources were taken into consideration.

A. Linux-VServer

Instead of using namespaces, Linux-VServer implements (through a patch) its own mechanisms in Linux kernel to provide process, network and CPU isolation. This is because Linux-VServer is the oldest implementation of container-based system for Linux and the namespace support into the kernel came up more recently. The system limits the scope of the file system from different processes through the traditional *chroot* system call and prohibits unwanted communications between them by using a technique called global PID space. The main benefits of this technique 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 to the impossibility to re-instantiate processes with the same PID [8].

Linux-VServer does not virtualize network subsystems. Rather all network subsystems (such as routing tables and IP tables) are shared among the containers and also with the host system. The drawback is that the containers are unable to bind sockets to a subset of host IP and to change their own route table and firewall rules, it needs to be made by the host administrator [8].

B. OpenVZ

Unlike Linux-VServer, OpenVZ is built on top of kernel namespaces, making sure that every container has its own isolated subset of resources. The system uses PID and IPC namespaces in order to accomplish isolation between processes from different contexts. OpenVZ also implements the network namespace. Moreover, it also provides different network operation modes, such as Route-based, Bridge-based and Physical-based. The main distinction between them lies at operation layer. While Route-based works in Layer 3 (network layer), Bridge-based works in Layer 2 (data link layer) and Physical-based in Layer 1 (physical layer). In the Physical-based mode, it is possible to assign a real network device (such as eth1) into a container, providing a better network performance [9].

C. LXC

LXC is the most recent Linux implementation. In the same way as OpenVZ, LXC uses kernel namespaces to guarantee isolation among containers. LXC implements PID, IPC, File System and Network namespaces. Furthermore, it also offers different types of network configurations which are: Route-based and Bridge-based. Resource management is only performed via cgroups [21]. With cgroups it is possible defining network configurations, limiting the CPU usage and accomplishing isolation among processes from different containers contexts. LXC adopts the CFQ scheduler by default to control the I/O operations. In most recent kernel versions is it possible to control the I/O bandwidth per container via the *blkio* controller.

IV. Experiments

Our experiments were conducted with the contemporary Linux implementations of container-based virtualization: Linux-VServer, OpenVZ and LXC. Such experiments were performed upon two well-known evaluation perspectives: micro- and macro-benchmarks. By micro-benchmarks it is possible to measure the performance of basic Hadoop components before evaluating the system as a whole. Having the micro-benchmarks results, the macro-benchmarks are executed by stressing the MR model and the whole system. In this sense, the results obtained from micro-benchmarks have a direct impact on macro-benchmarks, making them very useful for identifying bottlenecks and mainly the reasons for performance overheads throughout the system with a more in-depth analysis

Our testbed environment comprises of four identical nodes with two 2.27GHz processors (with 8 cores each), 8M of L3 cache per core, 16GB of RAM, one 146GB disk and one gigabit network adapter. Once each system have kernel requirements that differ between each other, we have taken care in compiling different kernels for each system and also in using the same release version. This ensures that the results are not influenced by increases and losses of performance introduced from distinct kernel releases. The kernel 2.6.32-28 was chose, as it has support to all systems'

patches and configurations. For OpenVZ, an additional patch (2.6.32-feoktistov) is needed to allow us to use namespaces and containers. Likewise, to put up the Linux-VServer, the patch (2.3.0.36.29.4) developed by the Linux-VServer team was installed into the kernel. Otherwise OpenVZ and Linux-VServer, LXC already comes available with the kernel mainline (since 2.6.24) and any modification was needed. In such a way, we just installed the tool kit (lxc-tool) required to manage the containers and ensure that all requirements extracted from *lxc-checkconfig* utility are met.

Finally, to analyze the behavior in using the MR model on the systems, we set up a Hadoop cluster, including the HDFS and the MR scheduler. The HDFS was configured by using 2 Namenodes and 4 Datanodes distributed across the cluster. The replication factor was set to 3 and the Java heap size was configured to be 1024MB. The number of Map and Reduce tasks per node was set to 6 and 2 respectively, in an attempt to balance the overall utilization across the 8 available cores. Finally, 30 minutes of task timeout was configured. All results were analyzed considering a confidence interval of 95%. The next sections presents the obtained results.

V. Micro-benchmarks

A. Performance Evaluation of HDFS

To identify the best results of performance for HDFS, we chose the TestDFSIO benchmark which comes bundled with the Hadoop distribution and give us the writing/reading performance corresponding with the throughput. TestDFSIO has been traditionally adopted by Hadoop users for identifying performance bottlenecks in networks, operating system and HDFS configurations, giving an insight of how efficient the cluster in terms of I/O is. The metric Throughput is measured in Mbps. For a job using N map tasks, the throughput is defined considering an index that goes from 1 to N where N denotes the number of tasks. It is represented as follows:

$$Throughput(N) = \frac{\sum_{i=0}^{N} filesize_i}{\sum_{i=0}^{N} time_i}$$

As the metric is based on the elapsed time for writing/reading files by individual tasks, it is easily observed that the size of such a file infers on the results. In that way, to obtain more accurate results we ran tests writing 10 files per MR cycle, varying the file size from 100 MB to 3000 MB. In an attempt to produce significant results when configuring replication factor for 2 and 3, we prefer to perform experiments with only write operations, since their results have a confidence interval close to 0, while we observed a confidence interval at an average of 50 for read operations. This decision allows a more fair evaluation between the systems.

The behavior of all the systems depict in Figures 1 and 2 tend to be linear when the size of the files goes beyond 2GB. It is due of the copies transferred across the DataNodes in order to meet the replication factor policy. Considering a replication factor of 3, for a write operation of 10 files of

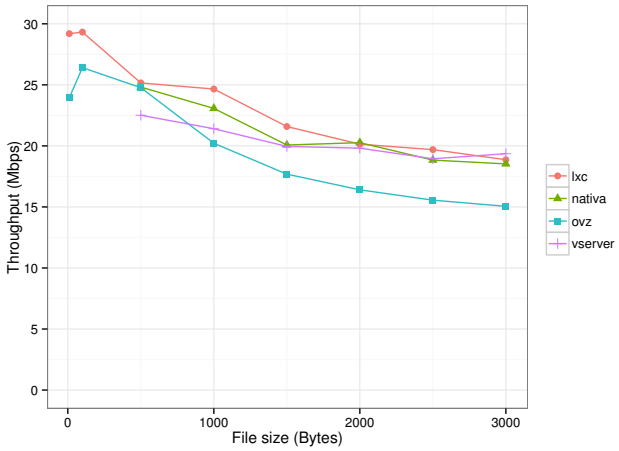

Figure 1. Performance evaluation of HDFS with a replication factor of 2

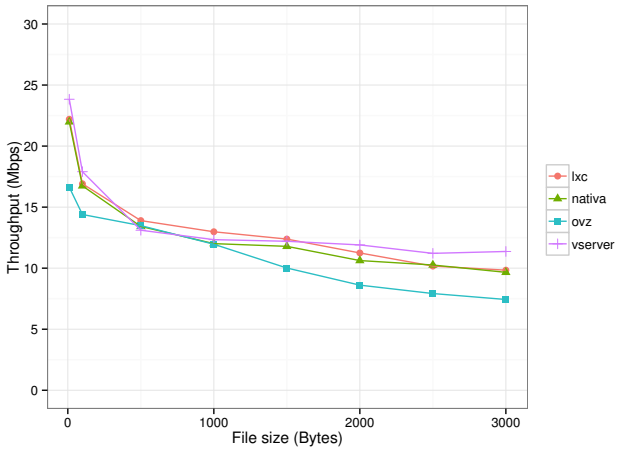

Figure 2. Performance evaluation of HDFS with a replication factor of 3

2GB, at least 60GB of data are transferred over the network. In such a way, we assume that the network throughput has influenced the results, since the capacity of the network does not reach its maximum until the data transferred reaches 60GB.

The result obtained from Linux-VServer in Figure 2 was similar to the results revealed in [7] when the network subsystem was stressed with the NetPipe benchmark [22]. Such behavior can be explained due to the different network implementation of each system. While OpenVZ and LXC were configured to take advantages of using the Bridge-based network in order to increase scalability, as described in Section III. Linux-VServer uses a Physical-based network which by its very nature reveals a latency very close to the native. However, with low scalability. That is why no significant difference was observed in the Linux-VServer while less data was transferred in the network, as shown

in Figure 1.

On the other hand, OpenVZ achieves a throughput at an average of 6.7Mbps for files of 3GB in size, which represents a loss of 3Mbps. It is due the default I/O scheduler adopted by the system. While LXC and Linux-VServer utilize a deadline scheduler for I/O operations that aggressively reorders requests to ensure improvement in I/O performance, Openvz uses CFQ scheduler in order to classify instructions by priorities, making the shared resources better distributed. Albeit there are benefits in using the CFQ scheduler in shared systems, the inherent overhead needs to be taken into consideration in container-based virtualization systems. Furthermore, considering we are using a single disk per node, the performance could be even better if the number of disks were increased.

B. NameNode Evaluation

We chose the NNBench benchmark in order to evaluate the NameNode component while observing its behavior when dealing with a large number of HDFS-related requests. NNBench comes packaged with Hadoop distribution. It was meant as a load test for simulating creating, reading, renaming and deleting files on HDFS. We performed two types of simulations for creating/writing and opening/reading. The first means that the files are first created and then written, while the second means that the files are first opened and then read.

The most remarkable metric which deserve a close inspection is the average latency, measured in milliseconds. NNBench was set up to generate operations on 1000 files on HDFS. At the end of each simulation it was possible realizing a number of 8000 operations carried out.

Table I
PERFORMANCE EVALUATION USING NNBENCH
(TIMES ARE REPORTED IN MS)

	Native	LXC	OpenVZ	VServer
Open/Read	0.51	0.52	0.51	0.49
Create/Write	54.65	56.89	51.96	48.90

The ability of the NameNode in dealing with both types of simulations was similar as indicated in Table I. However, the results become slightly significant when we analyze the differences between the systems. By observing the results revealed for creating/writing operations, Linux-VServer reaches a latency at a average of 48ms, while LXC obtained the worst result at an average of 56ms. The differences are not so significant if the numbers are considered. However, the strengths are that no exception was observed during the high HDFS management stress, and that all systems were able to respond effectively as the native.

C. MapReduce benchmark

The job turnaround time on high throughput MR clusters is tightly influenced by the ability of the job scheduler to handle requests as efficient as possible. MRBench is a benchmark that puts stress on the MR layer for the sole purpose of identifying its efficiency while dealing with several job requests. As such it comes with the Hadoop distribution and runs a job multiple times, taking an average of all runs.

Before beginning, MRBench just creates a text file containing a given number of lines of generated data. The number of lines we defined was in the order of 1 million. It is also possible defining the number of Map and Reduce tasks as configuration option. Due to our MR cluster characteristics, we defined 24 and 8 of Map and Reduce tasks respectively.

MRBench starts by taking text lines as input format, runs some processing on it and writes out data as text again. On the next step, the reduce function ignores the key and writes the values to the output again. It is worth noting that the benchmark does not put any stress test on HDFS, since no write operation is performed on the file system.

Table II
PERFORMANCE EVALUATION USING MRBENCH
(TIMES ARE REPORTED IN MS)

	Native	LXC	OpenVZ	VServer
Execution time	14251	13577	14304	13614

There is a certain similarity when the results are compared between the systems, as shown in Table II. No timeout was observed and all systems were able to respond during a sequence of 50 job requests. The results obtained from MRBench are useful and show that MR layer suffers no substantial effect while running on the different container-based virtualization systems.

VI. MACRO-BENCHMARKS

Macro-benchmarks are considered benchmarks that stress out many components of a system and can normally give more significant results, as it implicitly includes the components before evaluated by micro-benchmarks. The macro-benchmarks selected include: Wordcount [23], Terasort [24] and IBS [25].

A. Analyzing performance with Wordcount

WordCount is a simple application that counts the number of occurrences of each word in a given input dataset. It also comes with the Hadoop bundle and it is widely used as a way for comparing the performance among different Hadoop clusters. This experiment does not make any stress test, instead it only demonstrates a performance comparison when a real-world application is running on. The input file was created by looping a sample text file until it reach 30GB of size. We believe that such a dataset is large enough to find out the best results of performance, taking into account the characteristics of the cluster.

As can be observed in Figure 3, all container-based system achieved a near-native performance. We suppose that the peak of performance degradation from OpenVZ is

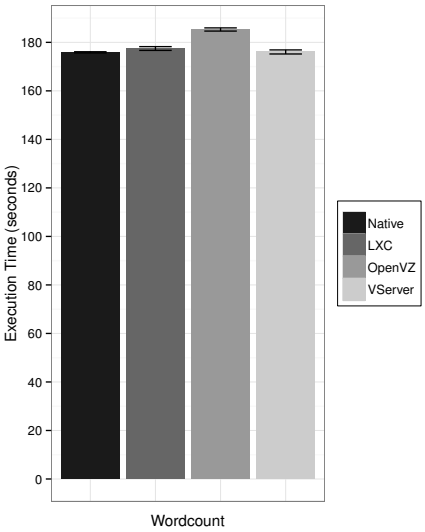

Figure 3. Performance evaluation using Wordcount

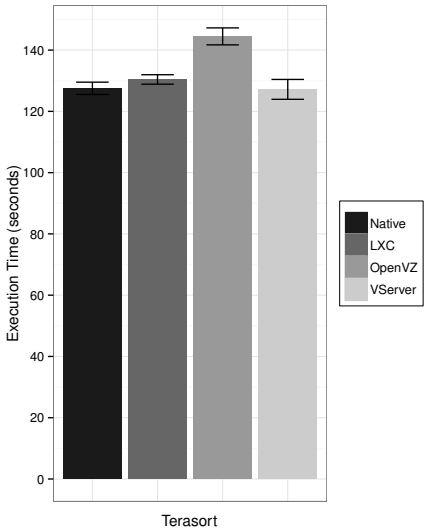

Figure 4. Performance evaluation using TeraSort benchmark

explained by the experiment early analyzed in Section V-A, while the I/O scheduler introduced some impact on HDFS performance.

B. Terasort benchmark

Terasort is the well-known benchmark to identify how fast a Hadoop cluster is. It also comes with the Hadoop distribution and has been adopted in many cases in the industry to put records in large-scale clusters.

Terasort is a standard map/reduce sort. The goal of the benchmark is to sort an amount of data as fast as possible. To do so, a execution of Terasort consists of at least 2 steps: generating the input data and running on such input data. There is a thirty step that validates the sorted output. Although we consider the latter, we decided not to put in the paper due to lack of space.

TeraGen was used to generate 30GB of random data to be conveniently used as input data for the subsequent Terasort. We consider a HDFS block size of 64MB to ensure that the time to start/stop map tasks are not larger than the time to perform the tasks, due of the characteristics of the cluster.

The behaviors of the systems depicted in Figure 4 were similar to the Wordcount and the reason we suppose to be the same—the HDFS overheads identified in Section V-A.

C. Performance Isolation

During the past few years, container-based virtualization was characterized as a virtualization architecture that does not promise a good performance isolation, as also revealed in our previews work [7]. Nowadays, with the advent of novel container-based technologies such as LXC, and with the improvements in cgroup feature to better control the limits of the system resources, the results of isolation are uncertain and deserve to be further studied. To do so we chose LXC as the representative of the container-based virtualization to be evaluated, as it is the most recent implementation and also because there has been a trend toward using it in MR clusters. Mesos, for instance, uses LXC with the purpose of separating the resources by isolating them from different frameworks like MPI and Hadoop. Likewise, YARN has also incorporated LXC and cgroups with the aim of isolate the resources such as memory per application.

The experiments were conducted by using the IBS benchmark on a modified Hadoop cluster to run two containers per node. We divide the overall available resources per node by half, including CPU, memory and disk I/O. In LXC, the cgroup feature is used to do so. The subsystem *cpuset.cpus* was used to restrict the number of CPU usage per container. The amount of available memory per container was configured to be 4GB by using the subsystem *memory.limit_in_bytes*. Finally, the maximum I/O bandwidth per container was limited by putting weights on the I/O blocks. The subsystem *blkio.weight* implements the block I/O controller. It was incorporated into the LXC since the 2.6.34 kernel release.

The evaluation of the performance isolation consists of two steps: (1) the execution time of a given baseline application running on one container is collected—we chose the Terasort; and (2) such application is rerun side-by-side with a stress test (CPU, memory, I/O, forkbomb) on another container. The metric is obtained by comparing the execution time from steps (1) and (2). At the end, it is possible observing how much the performance of the Terasort is impacted by different stress tests running side-by-side.

The results we obtained are shown in Table III. The numbers represent the performance degradation for each resource evaluated. It was possible realizing that the limits

Table III
PERFORMANCE ISOLATION OF DIFFERENT RESOURCE USING IBS BENCHMARK SUITE.

	CPU	Memory	I/O	Fork Bomb
LXC	0%	8.3%	5.5%	0%

of the CPU usage per container is working well and no significant impact was noted. However, during the memory and I/O stress tests, a little performance degradation needs to be taken into account. The result could be worse, since the I/O bandwidth was set to be distributed fairly among the containers.

Finally, the fork bomb stress test reveals that the LXC has a security subsystem that ensure feasibility, as the operating system process scheduling was not affected.

VII. RELATED WORK

The use of virtualization systems in resource-intensive environments has been the subject of several studies in the past few years due of its potential to improve the manageability and because of the performance overheads reduction with the improvement of hardware-assisted technologies, such as Intel-VT and AMD-V. Nevertheless, the results obtained so far have proved that virtualization systems are still unable to achieve the same native performance. Once hypervisors have the function of translating instructions from upper layers (virtual machines) to the lower layer (hardware), in most cases this is the major cause for the performance overheads.

One the other hand, there has been very few works exploring MR on virtualization systems. Such works have analyzed how virtualization systems can benefit MR clusters by improving reliability through fault tolerance techniques and what is the impact of virtualization on such clusters. Ibrahim et al. [6], for instance, measured and analyzed the performance of HDFS on both physical and virtual cluster—to do so, was used the hypervisor-based system Xen—when transferring data from and to HDFS. The results reveal that MR on virtual machines is unable to reach the same native performance. It is already expected due to the very nature of hypervisor-based systems. Moreover, the work also lacks a performance comparison between virtual and physical clusters using further virtualization systems involving others architectures.

CLOUDLET [26] is a implementation of MR on virtual machines that despite of the well-know I/O overheads, tries to overcome it by benefiting of the management features, increasing system reliability, achieving high throughput and execution accurateness. The use of container-based systems could help CLOUDLET by decreasing the performance overheads while keeping manageability.

VMWare has recently announced the Big Data extension to vSphere platform through the Serengeti project [27], which take advantages of virtualization capabilities to support Hadoop workloads. Results obtained from experiments reveal that the impact of virtualization was about 13% of performance overhead for Terasort [27]. The best Terasort result we obtained when comparing the container-based systems was an overhead at average of 2% for the LXC. YARN [14] is the next generation of Hadoop, which will use container-based virtualization to share a cluster among different applications. Mesos [13] is a platform that leverages containers to isolate resources among frameworks (such as Hadoop and MPI) that share partitions of a cluster. It is not clear why the authors chose LXC as the virtualization platform to do so. However, the results we obtained on this work demonstrate that there are substantial differences among the container-based systems.

Although some works are already taking advantages of container-based systems on MR clusters, there are still no work studying the impact of this type of virtualization in the performance of MapReduce applications. For the best of our knowledge, this is the first work to (1) evaluate the performance overhead of container-based systems on MR application, (2) compare all of recent container-based implementations in terms of performance and manageability for running MR clusters and (3) evaluate the performance isolation of LXC for MR applications.

VIII. CONCLUSION AND FUTURE WORK

This paper presented a performance comparison between the current container-based virtualization systems for MapReduce clusters. Furthermore, we also examined the performance isolation of the most recent container-based system, as it is in constantly evolving and because new capabilities have recently emerged.

We have pointed out usage cases where virtualization have been adopted in MapReduce clusters. In cloud computing environments, virtualization had an important contribution, by allowing server consolidation, more security and better resource utilization. The ability of cloud computing environments in supporting MapReduce workloads have grown in the last years due of the increase volume of data we face today and the high cost for computing such a large dataset in traditional clusters. On the other hand, traditional MapReduce clusters are typically shared among many users or institutes which may have different software requirements that differ between on another. As presented, Mesos and YARN are systems that benefits from using a container-based system to provide better resource sharing between programming frameworks, such as Hadoop and MPI.

In spite of these cases where virtualization is very useful for MapReduce clusters, their use is only feasible if the fundamental performance overhead is reduced. We found in this work that all container-based systems reach a near-native performance for MapReduce workloads while contributing with many management capabilities, such as performance isolation, checkpoint and live migration.

The results of performance isolation reveled that the LXC has improved its capabilities of restrict resources among containers. On our previews work [7], we found a poor security and performance isolation for LXC. Nowadays,

with the advent of new capabilities into LXC, such as the *blkio* controller, the resources could be better isolated when evaluated with a I/O intensive workload. This makes its use very attractive in MapReduce environments.

As future work, we plan to study the performance isolation at the network-level. We also plan to study the aspects regarding the green computing, such as the trade-off between performance and energy consumption while using container-based systems and hypervisor-based systems.

REFERENCES

[1] Hadoop, "Apache Hadoop Website," 2013, accessed on August 2013. [Online]. Available: http://hadoop.apache.org

[2] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash *et al.*, "Apache hadoop goes realtime at facebook," in *Proceedings of the 2011 international conference on Management of data*. ACM, 2011, pp. 1071–1080.

[3] "Amazon Elastic MapReduce (Amazon EMR)," 2013. [Online]. Available: http://aws.amazon.com/elasticmapreduce/

[4] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[5] 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," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[6] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi, "Evaluating mapreduce on virtual machines: The hadoop case," in *Cloud Computing*. Springer, 2009, pp. 519–528.

[7] M. Xavier, M. Neves, F. Rossi, T. Ferreto, T. Lange, and C. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, 2013, pp. 233–240.

[8] B. des Ligneris, "Virtualization of linux based computers: the linux-vserver project," in *High Performance Computing Systems and Applications, 2005. HPCS 2005. 19th International Symposium on*. IEEE, 2005, pp. 340–346.

[9] "OpenVZ," 2013. [Online]. Available: http://www.openvz.org

[10] "Linux Containers," 2013. [Online]. Available: http://lxc.sourceforge.net

[11] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra, and S. Huss-Lederman, *MPI: the complete reference*. MIT press, 1995.

[12] M. V. Neves, T. Ferreto, and C. Rose, "Scheduling MapReduce Jobs in HPC Clusters," in *Euro-Par 2012 Parallel Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 179–190.

[13] 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*. USENIX Association, 2011, pp. 22–22.

[14] YARN, "Apache Hadoop NextGen MapReduce (YARN)," 2013, accessed on August 2013. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[15] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[16] D. Borthakur, "The Hadoop Distributed File System: Architecture and Design," Jan 2007, accessed on August 2013. [Online]. Available: http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf

[17] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: A Runtime for Iterative MapReduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ACM. New York, New York, USA: ACM Press, 2010, pp. 810–818.

[18] Spark, "Spark Website," 2013, accessed on August 2013. [Online]. Available: http://spark-project.org.

[19] T. White, *Hadoop: the definitive guide*. O'Reilly, 2012.

[20] E. Biederman and L. Networx, "Multiple instances of the global linux namespaces," in *Proceedings of the Linux Symposium*. Citeseer, 2006.

[21] "Control groups definition, implementation details, examples and api," 2013. [Online]. Available: http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[22] Q. O. Snell, A. R. Mikler, and J. L. Gustafson, "Netpipe: A network protocol independent performance evaluator," in *IASTED International Conference on Intelligent Information Management and Systems*, vol. 6, 1996.

[23] "Wordcount benchmark," 2013. [Online]. Available: http://wiki.apache.org/hadoop/WordCount

[24] O. OMalley, "Terabyte sort on apache hadoop," *Yahoo, available online at: http://sortbenchmark. org/Yahoo-Hadoop. pdf,(May)*, pp. 1–3, 2008.

[25] 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*. ACM, 2007, p. 6.

[26] S. Ibrahim, H. Jin, B. Cheng, H. Cao, S. Wu, and L. Qi, "Cloudlet: towards mapreduce implementation on virtual machines," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*. ACM, 2009, pp. 65–66.

[27] "Serenget vmware project," 2013. [Online]. Available: http://www.vmware.com/br/hadoop/serengeti.html