Evaluating Virtualization for Hadoop MapReduce on an OpenNebula Cloud

Pedro Roger Magalhães Vasconcelos, Gisele Azevedo de Araújo Freitas
Federal University of Ceará
Ceará, Brazil

Abstract

Cloud computing provides access to a set of resources such as virtual machines, storage and network as services. In this context, virtualization has been used as a platform for resource-intensive applications, like Hadoop, as it has brought features like server consolidation, scalability and better resources usage. OpenVZ and KVM are very popular and widely used virtualization platforms with distinct virtualization modes: container-based and full-virtualization. In this work, we conducted benchmarks to measure the performance of a Hadoop cluster deployed on OpenNebula clouds with KVM and OpenVZ. Our results show that OpenVZ performs better than KVM in the CPU and I/O reading benchmarks. KVM achieves better performance in the I/O writing benchmarks.

1. Introduction

Cloud Computing is creating an environment suitable for web services. Nowadays, the most common architecture is distribute data and computation on different geographic areas to a centralized cloud computing architecture, where the computations and data are operated somewhere in the cloud, where data centers are owned and maintained by third party. However, in terms of resources, the three main characteristics of cloud are [1]: on-demand unlimited data storage, on-demand computation power with no lock, mainly represented as Virtual Machines (VMs), and using internet, limited bandwidth connection, to access, use and process these resources.

Typically a cloud is hosted on a server farm with a large amount of clustered computers. These provide the hardware resources. The cloud provider (the organization that hosts the servers) offers an interface for users to pay for a certain amount of processing power, storage or computers in a business model. These resources can then be increased or decreased based on demand, so the user only needs to focus on it's contents whereas the provider takes care of maintenance, security and networking [2].

The servers in the server farm are usually virtualized, although they are not required to be to actually be included in a cloud. Virtualization is a ground pillar in cloud computing; it enables the provider to maximize the processing power on the raw hardware and gives the cloud elasticity, the ability for users to scale the instances required. It also helps providing two other key features of a cloud: multitenacity, the sharing of resources, and massive scalability, the ability to have huge amounts of processing systems and storage areas (tens of thousands of systems with large amounts of terabytes or petabytes of data) [3].

Virtualization is a logical representation of the computer using software. In a physical computer, there is a single operating system running one or more applications. Using virtualization, a software that runs on a single physical computer consist by physical resources running into different virtual machines. Hence several operating systems can be run inside a single machine. A result from a virtual computing does not affect other virtual computing. The main advantage of virtualization is the effective use of hardware. That is, sharing resources to each virtual machine, there has been complete utilization of the hardware. Billions of dollars have been invested on the research on controlling heat dissipation in data center. The only way is to use the less number of servers, hence virtualization on server allows less physical hardware and less dissipation of heat. Nowadays, there has been a significant increase in the cost of the hardware, hence virtualization allows fewer physical hardware and hence reduced cost. Some of the parameters which adds up to the cost saving are easier maintenance and less electricity. Redeployment and backups are made easier in virtualization by using a snapshot mechanism [4].

Nowadays, there are lot of an open source cloud computing solutions for providing infrastructure environments that include public, private or hybrid clouds such as Eucalyptus, OpenNebula, and CloudStack. OpenNebula [5] is an open source solution that allows easily deploys private/hybrid infrastructure clouds based on IaaS model. All the physical hosts on the cloud do not need to have homogeneous configuration, but it is possible to use different hypervisors on different GNU/Linux distributions on a single OpenNebula cluster.

OpenNebula has a great flexibility regarding hypervisor usage since it natively supports KVM/Xen (which are open source) and VMware ESXi. Drivers provided by the OpenNebula

community provide support for OpenVZ operating system-level virtualization.

The Kernel-based Virtual Machine (KVM) is a full-virtualization solution for the Linux kernel. KVM requires a processor with hardware virtualization extension. In full virtualization, a layer, commonly called the hypervisor or the virtual machine monitor, exists between the virtualized operating systems and the hardware. This layer multiplexes the system resources between competing operating system instances. The full virtualization has got the advantage to integrate physical machines with different characteristics for example Intel x86 with AMD64 without doing any modification in OS kernel.

A hypervisor is composed of several components usually having to write a scheduler, memory management, I/O-stack for a new hypervisor, as well as drivers for the architecture on which the hypervisor is targeted at. KVM unlike other hypervisors such as Xen, has focused the implementation on the guest handling of the virtualization, letting the Linux kernel operate as the hypervisor. Since Linux has developed into a secure and stable operating system, as well as having some of the most important features for a hypervisor, such as a scheduler and a plethora of drivers, it is more efficient to reuse and build upon this rather than reinvent a hypervisor [6].

KVM consists of two components [7]: kernel module and user-space. Kernel module sees to the virtualization of hardware resources through /dev/kvm and kill command. With /dev/kvm, guest operating system may have its own address space allocated by the Linux scheduler. The physical memory mapped for each guest operating system is actually the virtual memory of its corresponding process. A set of shadow page tables is maintained to support the translation from guest physical address to host physical address. User space takes charge of the I/O's virtualization by employing a lightly modified QEMU to simulate the behavior of I/O or sometimes necessarily triggering the real I/O. KVM also provides a mechanism for user-space to inject interrupts into guest operating system. Any I/O requests of guest operating system are trapped into user-space and simulated by QEMU.

OpenVZ (Open Virtuozzo) is an operating system- level virtualization technology based on the Linux kernel and operating system. OpenVZ allows a physical server to run multiple isolated operating system instances, known as containers, Virtual Private Servers (VPSs), or Virtual Environments (VEs).

OpenVZ includes a custom kernel and several user-level tools [7]. The custom kernel is a modified Linux kernel with the function of virtualization, isolation, checkpointing and resource management. The resource manager comprises three components: fair CPU scheduler, user beancounters and two- level disk quota. OpenVZ implements a two-level CPU scheduler: the OpenVZ scheduler determines the CPU time slice's assignment based on each container's cpu unit value, and the Linux scheduler decides the process to execute in this container with the standard Linux process priority. Real CPU time is distributed proportionally. OpenVZ controls container operations and system resources such as memory, internal kernel objects and network buffers with about 20 parameters in user beancounters. These resources can be changed without container rebooting [7].

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 ta- bles, firewall rules and so on. It also provides some network operation modes, such as Route-based, Bridge-based and Real Network based [8].

Since the OS-level virtualization works at the operating system level, all virtual instances share a single operating system kernel. For this reason, OS-level virtualization is supposed to have a weaker isolation when compared to hypervisor-based virtualization [8]. The physical server and single instance of the operating system is virtualized into multiple isolated partitions, where each partition replicates a real server. The OS kernel will run a single operating system and provide that operating system functionality to each of the partitions.

While hypervisor-based virtualization provides abstraction for full guest OS's (one per virtual machine), operating system-level virtualization works at the operation system level, providing abstractions directly for the guest processes. In practice, hypervisors work at the hardware abstraction level and operating system-level virtualization at the system call layer [9].

This paper evaluates the performance of a Hadoop MapReduce cluster on an OpenNebula cloud under two different types of virtualization: full virtualization and operating system-level virtualization. Results of various Hadoop benchmarks under these virtualization types are presented here.

The rest of the paper is organized as follows. Section 2 discusses the MapReduce model and the Hadoop framework. Section 3 discusses the related works. Section 4 presents the experiment setup and the benchmarking tools. Section 5 presents and discusses the results of the experiment. Finally, Section 6 concludes the study.

2. MapReduce

MapReduce (MR) [9.] is a programming model that works on large datasets. Many organizations use MapReduce model for computing when they have huge datasets and need to process them within short time. The main idea of the MapReduce model is to hide details of parallel execution and allow users to focus only on data processing strategies. MapReduce [10] performs a series of operations on the input key and value. MapReduce works by breaking the processing into two phases: the map phase and the reduce phase [11]. So, the MapReduce model consists of two primitive functions: Map and Reduce. In MapReduce, the Map function processes the input in the form of key/value pairs and generates intermediate key/value pairs, and the Reduce function process all intermediate values associated with the same intermediate key generated by the Map function [11]. Developers specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

The Map function produces a set of intermediate key/value pairs, the MapReduce library groups together all intermediate values associated with the same intermediate key 1 and passes them to the Reduce function. The input for MapReduce is a list of (key 1, value 1) pairs and Map() is applied to each pair to compute intermediate key-value pairs, (key 2, value 2). The Reduce function accepts an intermediate key 1 and a set of values for that key. It merges together these values to form a possibly smaller set of values. It means for each key 2, Reduce() works on the list of all values, then produces zero or more aggregated results.

The intermediate key-value pairs are then grouped together on the key-equality basis, i.e. (key 2, list (value 2)).

Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows handling lists of values that are too large to fit in memory.

Many real world tasks are expressible in this model. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Apache Hadoop is an open-source Java implementation of MapReduce [11]. Hadoop consists of two layers: a data storage layer called Hadoop DFS (HDFS) and a data processing layer called Hadoop MapReduce Framework. HDFS is a block-structured file system managed by a single master node like Google's GFS. Each processing job in Hadoop is broken down to as many Map tasks as input data blocks and one or more Reduce tasks.

In Facebook, event logs from its website are imported into a 600-node Hadoop data warehouse, where they are used for a variety of applications, including business intelligence, spam detection, and ad optimization. The warehouse stores 2 PB of data, and grows by 15 TB per day. In addition to production jobs that run periodically, the cluster is used for many experimental jobs, ranging from multi-hour machine learning computations to 1-2 minute ad-hoc queries submitted through an SQL interface to Hadoop called Hive. The system runs 7500 MapReduce jobs per day and is used by 200 analysts and engineers [12].

As many companies now adopt Hadoop, the requirements of what input varies from flat files to databases. In a database, one can take an example of rows from the table.

Map functions are distributed between all nodes in cluster and locality of the data can be considered unknown. There are also better ways to balance the data and jobs that are local to the data.

2.1. Hadoop MapReduce and Hadoop Distributed File System (HDFS)

Hadoop MapReduce is a distributed programming framework and an execution environment for MapReduce programs [9]. Hadoop is heavily used by companies such as Yahoo!, Facebook and eBay to perform thousands of computations per day over petabytes of data [13] The execution environment also includes a job scheduling system that coordinates the execution of multiple MapReduce programs, which are submitted as batch jobs. A MapReduce job consists of multiple map and reduce tasks that are scheduled to run in the Hadoop cluster 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.

Hadoop splits are fixed-size, whereas a separate Map task is created for each split. The default Hadoop MapReduce split size is the same as the default size of an HDFS block, which is 64 MB. Hadoop performs data locality optimization by running the Map task on the node where the input data resides in the HDFS. With the default HDFS replication factor of three, files are concurrently stored on three nodes; hence, splits f the same file can be concurrently processed on three nodes without the need for being copied before.

Job scheduling in Hadoop is performed by a master, which manages a number of slaves. Each

slave has a fixed number of map slots and reduce slots in which it can run tasks. Typically, administrators set the number of slots to one or two per core. The master assigns tasks in response to heartbeats sent by slaves every few seconds, which report the number of free map and reduce slots on the slave. Hadoop's default scheduler runs jobs in FIFO order, with five priority levels. When the scheduler receives a heartbeat indicating that a map or reduce slot is free, it scans through jobs in order of priority and submit time to find one with a task of the required type [12]. The Hadoop Distributed File System (HDFS) [10] is the main storage system used by Hadoop. HDFS is a block-structured file system managed by a single master node. Each processing job in Hadoop is broken down to as many Map tasks as input data blocks and one or more Reduce tasks. HDFS creates multiple replicas of data blocks and distributes them among the cluster nodes. HDFS maps all the local disks to a single file system hierarchy allowing the data to be dispersed at all the data/computing nodes. HDFS also replicates the data on multiple nodes so that a failure of nodes containing a portion of the data will not affect computations, which use that data. Hadoop schedules the MapReduce computation tasks depending on the data locality and hence improving the overall I/O bandwidth. This setup is well suited for an environment where Hadoop is installed in a large cluster of commodity machines.

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 [9].

The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later [14].

Hadoop stores the intermediate results of the computations in local disks, where the computation tasks are executed, and then informs the appropriate workers to retrieve (pull) them for further processing. Although this strategy of writing intermediate result to the file system makes Hadoop a robust technology, it introduces an additional step and a considerable communication overhead, which could be a limiting factor for some MapReduce computations. Different strategies such as writing the data to files after a certain number of iterations or using redundant reduce tasks may eliminate this overhead and provide a better performance for the applications. The placement of replicas is critical to HDFS reliability and performance. Optimizing replica placement distinguishes HDFS from most other distributed file systems. This is a feature that needs lots of tuning and experience. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization.

Large HDFS instances run on a cluster of computers that commonly spread across many racks. Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.

To minimize global bandwidth consumption and read latency, HDFS tries to satisfy a read request from a replica that is closest to the reader. If there exists a replica on the same rack as the reader node, then that replica is preferred to satisfy the read request. If a HDFS cluster spans multiple data centers, then a replica that is resident in the local data center is preferred over any remote replica [14].

It is a distributed file system with similarities to other file systems, but with some differences. The HDFS is highly fault tolerant and is designed to run on low-cost hardware. Provides high performance processing large volumes of data. From end user perspective, the HDFS is seen as a traditional file system.

3. Related Work

Virtualization is getting more and more popular in a cloud environment, an example is the Amazon elastic MapReduce, [13]. A cloud based MapReduce [13] is offered by the Amazon web services and it is called as Elastic MapReduce (EMR). Elastic MapReduce is a web service to which users submit MapReduce jobs. The service takes care of provisioning resources, configuring and tuning Hadoop, staging data, monitoring job execution, instantiating new virtual machines in case of failure, etc. However, this service has a number of limitations. First, it is restricted to Amazon EC2 resources. Users are not able to use Elastic MapReduce with resources from other public clouds or from private clouds, which may be less expensive or even free of charge. This is especially true for scientists who have access to clouds administrated by their institution and dedicated to scientific computing. This EMR allows user to sign up to the Amazon web service and after getting sign in, user can submit their MapReduce jobs using EMR API

which is developed by some programming models like python or java. These MapReduce jobs are then sent to the Hadoop cluster, which consist of three virtual machines. That is, Unique master VM, which acts as HDFS and schedules MapReduce task over other VMs, Multicore VMs which produces storage for HDFS and computes all the MapReduce tasks. Finally a multitask VMs which don't store any data but executes MapReduce task. Moreover, Elastic MapReduce is provided for an hourly fee, in addition to the cost of EC2 resources. This fee ranges from 17% to 21% of the price of on-demand EC2 resources. It is impossible to use a different virtual machine image than the one provided by Amazon. Finally, some users may have to comply with data regulation rules, forbidding them from sharing data with external entities like Amazon.

In [10] the authors analyzed the performance of the K-Means Clustering Algorithm when running Hadoop MapReduce on Eucalyptus. Several tools like Ganglia, TestDSFIO.java, and others Linux performance measuring tools have been used to measure the performance. The paper discusses how the performance of K- Means Clustering Algorithm scales up with number of nodes on Eucalyptus cloud. Their results show a good performance of a MapReduce job running in four nodes. They noted that increase in number of nodes boosted performance significantly, and also the performance gain increases as number of nodes scale up.

The authors in [1] discuss the performance between a physical Hadoop cluster and a virtualized one. The performance gap in WordCount and HDFS benchmarks is markedly increases as the data size is increasing in both cases writing data to or reading data from the HDFS. The authors draw several open issues and explanations about the poor performance due the I/O competing when VMs are deployed within same physical node.

In an effort to minimize the performance degradation for virtualized MapReduce workloads, the authors in [15] presented a scheduler that implements three mechanisms to improve the efficiency and fairness of the existing VMM scheduler (Xen). First, the characteristics of MapReduce workloads facilitate batching of I/O requests from VMs working on the same job, which reduces the number of context switches and brings other benefits. Second, because most MapReduce workloads incur a significant amount of I/O blocking events and the completion of a job depends on the progress of all nodes, the authors proposed a two-level scheduling policy to achieve proportional fair sharing across both MapReduce clusters and individual VMs. Finally, the proposed scheduler also operates on symmetric multi-processor (SMP) enabled platforms. The key to these improvements is to group the scheduling of VMs belonging to the same MapReduce cluster.

In [16], Chen He, Derek Weitzel, David Swanson and Ying Lu discussed experiments by running Hadoop MapReduce on a grid which provides a free, elastic, and dynamic MapReduce environment on the opportunistic resources of the grid. For Hadoop evaluation, they successfully extended Hadoop to 1100 nodes on Grid. From the evaluation, Chen and his team found that the unreliability of the grid makes Hadoop on the grid very challenging. K-means Clustering Using Hadoop Map Reduce by Grace Nila Ramamoorthy [17] evaluated the performance of K-Means clustering using Hadoop Map Reduce on standalone servers infrastructure. This model works well, but when it comes to scalability and efficient use of resources and need for rebuilding environment for different projects it becomes complex and very time consuming.

Many papers have studied the performance overheads of virtualization technologies, a few of them focusing on HPC and cloud computing environments

The work in [18] discuss the differences between the KVM and OpenVZ virtualizations in two open source cloud virtualization platforms: OpenNode and Proxmox VE. The two platforms are highly identical, but they are based on distinct Linux distributions, and there are minor differences. They allow systems administrators to easily deploy and manage virtualized infrastructure based on industry standard components (RHEL/CentOS, Debian GNU/Linux, libvirt, python, etc.).

However the two platforms are similar in the main features, OpenNode cannot reach the popularity of Proxmox VE. Although OpenNode has achieved the level of services provided by Proxmox VE, does not gain the desired popularity yet.

Proxmox VE uses unique virtualization API but OpenNode based on libvirt which supports several types of virtualization solution. This feature would be benefit for OpenNode in the future.

A study in [8] conducted a number of experiments in order to perform an in-depth performance evaluation of container-based virtualization for High Performance Computing. The authors chose Linux VServer, OpenVZ and Linux Containers (LXC) as virtualization technologies and performed a series of benchmarks like Linpack, Stream, NetPIPE, NPB and the Isolation Benchmark Suite. The authors believe that, 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, they found that all container-based systems have a near-native performance of CPU, memory, disk and network. The main differences between them lie in the resource management implementation, resulting in poor isolation and security.

In addition to open source virtual machine monitors comparison, the authors in [7] measure and

analyze the performance of three open source virtual machine monitors: OpenVZ, Xen and KVM. Through measuring and analyzing those virtualization platforms with SPEC CPU2006, LINPACK, Kernel compiling, RAMSPEED, LMbench, IOzone, Bonnie++, NetIO, WebBench, SysBench and SPEC JBB2005, they found that OpenVZ has the best performance and Xen follows OpenVZ with a slight degradation in most experiments, while KVM has apparently lower performance than OpenVZ and Xen. Furthermore, this indicates that operating system-level virtualization and para-virtualization have some apparent advantage on data-intensive applications such as disk I/O, net I/O, web server and database server.

However, operating system-level virtualization and para-virtualization have their own drawbacks: Operating system-level virtualization can only virtualize guest operating system that the kernel is the same as host operating system, and para-virtualization requires changes to the kernel of guest operating system. On the contrary, full-virtualization presents users the most convenient use without any modification to guest operating system.

4. Experimental Evaluation

To evaluate the performance of Hadoop cluster on each virtualization platform, we propose the establishment of two private OpenNebula clouds. The execution environments consists of two identical servers IBM BladeCenter HS21 with two Intel Xeon CPUs E5-2620 of 2.00GHz (with 6 cores and HT technology in each), 48GB of RAM, connected to a SAN via Fibre Channel, and interconnected through a Gigabit Ethernet network adapter and switch. The hosts systems were installed on an Ubuntu GNU/Linux 14.04.1 LTS amd64 system. The OpenNebula version used was 4.8.0.

Each Hadoop cluster consists of six virtual machines, one master and five slaves, with the same configuration: Ubuntu GNU / Linux amd64 LTS 12.04, 2 vCPUs, 2GB of vRAM, 1GB swap and 10GB of disk. The ext4 filesystem was used in all virtual machines.

The version of QEMU/KVM is 2.0.0. For best performance, virtual machines used KVM VirtIO paravirtualized drivers for disk and network. For OpenVZ, the kernel used was 2.6.32-openvz-amd64-042stab093.5.

We used Hadoop version 1.2.1 and Oracle Java version 1.7.0 45 in all virtual machines. The HDFS file system had 60GB (6 x 10GB per virtual machine) of distributed storage, HDFS block size was 64MB and replication factor was set to 3. The task timeout was set to 30 minutes. The following sections describe the benchmarks.

4.1. Wordcount

The WordCount program is an application that reads text files as input and computes the number of occurrences of each word in a file. The output file is a text file where each line contains a word and the number of times it occurred, separated by a tab.

Each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and 1. Each reducer sums the counts for each word and emits a single key/value with the word and sum. As an optimization, the reducer is also used as a combiner on the map outputs. This reduces the amount of data sent across the network by combining each word into a single record [19].

The WordCount already comes with the Hadoop default installation and is widely used as a method of comparing performance between different Hadoop clusters.

The input files used in the benchmark were generated from the concatenation of random text files downloaded from Project Gutenberg to the desired sizes: 64MB, 128MB, 256MB, 512MB, 1GB and 2GB.

As can be observed in Fig. 1, OpenVZ reached the lowest execution time of WordCount.

4.2. TeraSort

The goal of TeraSort benchmark is to sort certain volume of data as quickly as possible. It is a benchmark that combines the use of HDFS layer and MapReduce layer.

TeraSort consists of 3 map/reduce applications:

• TeraGen is a map/reduce program to generate the data.

• TeraSort samples the input data and uses map/reduce to sort the data into a total order.

• TeraValidate is a map/reduce program that

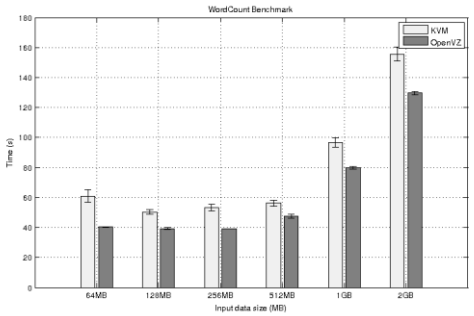

Figure 1: WordCount results

validates the output is sorted.

TeraGen generates output data that is byte for byte equivalent to the C version including the newlines and specific keys. It divides the desired number of rows by the desired number of tasks and assigns ranges of rows to each map. The map jumps

the random number generator to the correct value for the first row and generates the following row [20].

TeraSort is a standard map/reduce sort, except for a custom partitioner that uses a sorted list of N-1 sampled keys that define the key range for each reduce. In particular, all keys such that sample[i-1] ≤ key < sample[i] are sent to reduce i. This guarantees that the output of reduce i are all less than the output of reduce i+1. To speed up the partitioning, the partitioner builds a two level trie that quickly indexes into the list of sample keys based on the first two bytes of the key. TeraSort generates the sample keys by sampling the input before the job is submitted and writing the list of keys into HDFS. The input and output format, which are used by all 3 applications, read and write the text files in the right format. The output of the reduce has replication set to 1, instead of the default 3, because the contest does not require the output data be replicated on to multiple nodes [20].

TeraValidate ensures that the output is globally sorted. It creates one map per a file in the output directory and each map ensures that each key is less than or equal to the previous one. The map also generates records with the first and last keys of the file and the reduce ensures that the first key of file i is greater that the last key of file i-1. Any problems are reported as output of the reduce with the keys that are out of order [20].

We used variable sizes for generation of input data through TeraGen: 512MB, 1GB and 2GB.

The Figure 2 shows the performance of the two clouds in the execution of a variable size TeraSort run.

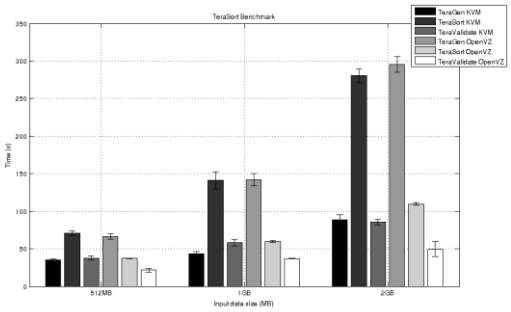

Figure 2: TeraSort results

4.3. TestDFSIO

The TestDFSIO benchmark is a read/write test to HDFS. It is useful to perform stress testing in the HDFS, to test parameters in NameNodes and DataNodes, to find bottlenecks and evaluate the cluster I/O rate. The TestDFSIO consists of two tests: the first, generates and writes the files in a HDFS directory; the second reads the files created by the first run and performs measurements.

Each file is read or written in a separate map task, and the output of the map is used for collecting statistics relating to the file just processed. The statistics are accumulated in the reduce to produce a summary.

In our runs, we inform the TestDFSIO to create 10 files of 100MB in HDFS. Fig. 3 demonstrates the performance of the two platforms during the benchmark. As we see, OpenVZ performance in writing tests was much lower than KVM. Although, in the reading tests OpenVZ performs better than KVM.

4.4. NNBench

The NNBench is useful for testing the hardware and configuring the NameNode. This test generates various requests to the HDFS with normally very small payloads for the purpose of stressing it. The benchmark can simulate requests for creating, reading, renaming and deleting files on HDFS. In our implementation, NNBench created 1000 files using 6 maps and 2 reducers. Figure 4 describes the NNBench runtimes for each cloud.

4.5. MRBench

The MRBench loops a small job a number of times. As such it is a very complimentary benchmark to the large-scale TeraSort benchmark suite because MRBench checks whether small job runs are responsive and running efficiently in the cluster. It put its focus on the MapReduce layer, as its impact on the HDFS layer is very limited.

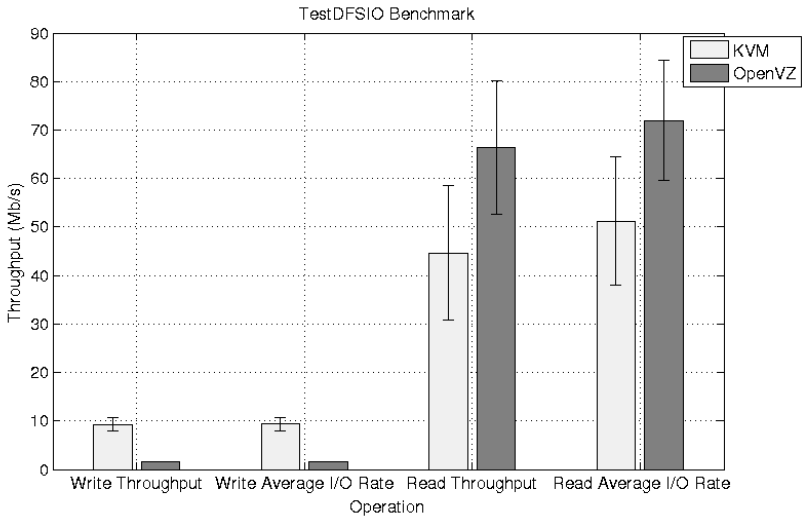

Figure 3: TestDFSIO results

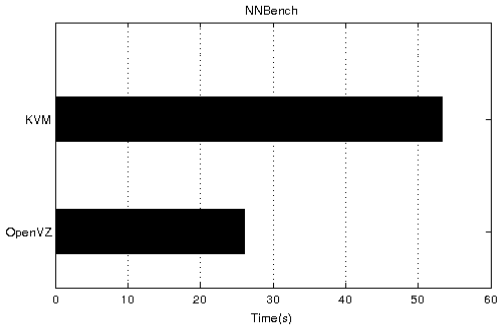

Figure 4: NNBench results

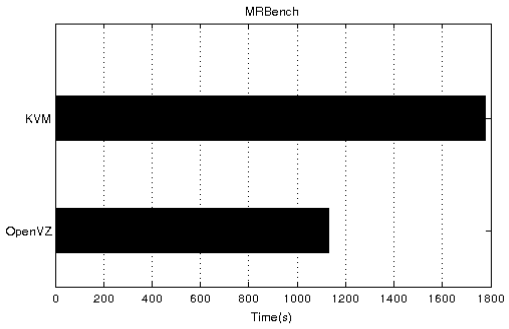

Figure 5: MRBench results

We ran MRBench in order to run a loop of 50 small test jobs. Fig. 5 shows the elapsed time for each one of the clouds.

4.6. Pi

The Hadoop Pi benchmark is a map/reduce program that estimates Pi using monte-carlo method. This benchmark is focused on computation and involves nearly no storage I/O or network traffic. Fig. 6 shows the Pi execution times for runs that calculates 10 billion samples spread across 6 maps tasks.

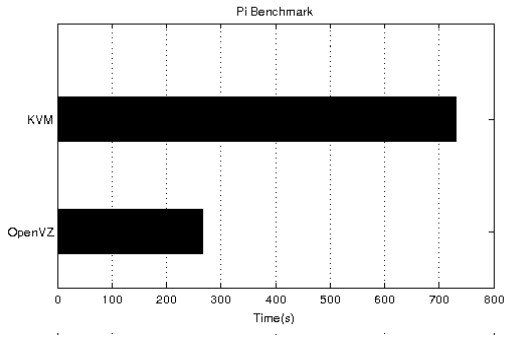

Figure 6: Pi benchmark results

5. Discussion

From the benchmarks results we can observe that OpenVZ performs better than KVM on CPU intensive tests like WordCount, TeraSort, TeraValidate, MRBench and Pi. OpenVZ too reaches better results than KVM on I/O reading tests as showed in the values of Read Throughput and Read Average I/O Rate operations of TestDFSIO benchmark. However, OpenVZ performs worst than KVM in I/O writing tests of large files in TeraGen and reached low rates of Write Throughput and Write Average I/O in TestDFSIO. But, in the sequential creation of innumerous small files in NNBench test the time elapsed in KVM run was almost twice of OpenVZ time.

In order to investigate the performance overhead between the two virtualization approaches we decided to run a series of microbenchmarks on these platforms.

5.1. Disk I/O

We ran the bonnie++ tool in both OpenVZ and KVM nodes. Bonnie++ is a program to test hard drives and file systems for performance or the lack thereof. There are a many different types of file

system operations, which different applications use to different degrees. Bonnie++ tests some of them and for each test gives a result of the amount of work done per second and the percentage of CPU time this took. For performance results higher numbers are better, for CPU usage lower are better

Bonnie++ benchmarks three things: data read and write speed, number of seeks that can be performed per second, and number of file metadata operations that can be performed per second. Metadata operations include file creation and deletion as well as getting metadata such as the file size or owner (the result of a fstat call).

The Table 1 describes the output of the Block I/O Bonnie++ test.

Table 1: Block I/O operations (KB/s)

Test	OpenVZ	KVM
Sequential Write	149720	251367
Rewrite	152056	131860
Sequential Read	4737075	217925

As we can see, OpenVZ performed much better than KVM in the Sequential Read test and KVM reach better rates of Sequential Write than OpenVZ. These tests comprove the better performance of OpenVZ in the reading and the good writing rates of KVM in tests executed in the TestDFSIO benchmark.

The bonnie++ suite calculated the average rate of metadata operations. The operations include the, random and sequential, creating and deleting of small files spread in the file system. The Table 2 exposes these results.

Table 2: File metadata operations (files/s)

Test	OpenVZ	KVM
Sequential Create	62994	60154
Sequential Delete	31057	3460
Random Create	63936	25944
Random Delete	17234	2194

The performance of KVM in the random creating and random deleting tests was much higher than OpenVZ. Despite this, KVM reaches higher rates in the sequential create benchmark. That can explain the better performance of KVM in the creation of small files in the NNBench benchmark.

We can see too that the KVM rate of sequential deleting is much above the OpenVZ's one. However this operation was barely used in the MapReduce benchmarks.

Table 3: CPU usage on disk benchmark (%)

Test	OpenVZ	KVM
Sequential Write CPU	29	83
Rewrite CPU	15	48
Read CPU	99	32
Random Seek CPU	33	54

The Table 3 demonstrates the CPU usage by each platform in the disk tests. OpenVZ uses less CPU than KVM in almost all tests (sequential write, rewrite and random seek), except in the disk reading benchmark.

5.2. Network performance

The network was intensely used during the benchmarks by both MapReduce and HDFS tasks. To separately inspect the network performance we use the Iperf tool. Iperf is a tool to measure maximum TCP bandwidth, allowing the tuning of various parameter. Iperf reports bandwidth, delay jitter, datagram loss.

Table 4: Network performance (Mbps)

Seconds	OpenVZ	KVM
1	940	954
2	938	944
3	940	934
4	940	944
5	940	934
6	940	947
7	938	932
8	940	943
9	940	947
10	940	932

The Table 4 shows the output of a 10 seconds Iperf run. We can observe that both KVM and OpenVZ reachs near native and stable rates.

5.3. CPU

To discover any difference in vCPU performance, we run the Systester benchmark utility.

This utility is a system stressing and benchmarking tool to test the system's stability by calculating up to 128 millions of Pi digits.

In our runs, we inform systester to calculate 8 millions of Pi digits.

Table 5: Computing of pi digits (seconds)

Pi digits	OpenVZ	KVM
1M	38.987	39.793
2M	40.814	41.615
4M	42.570	43.394
8M	43.795	44.614

The Table 5 shows that OpenVZ reaches 8 millions of Pi digits in less time. This can help understand the better performance of OpenVZ in CPU intensive tasks in the MapReduce benchmarks.

6. Conclusion

In cloud computing environments, virtualization had an important contribution, by allowing server consolidation and ease of deploy and management. The ability of cloud computing environments in supporting workloads have grown in the last years due of the advances in hypervisors and hardware's improves like virtualization-specific instructions sets.

A cloud can be 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.

KVM and OpenVZ are widely used solutions for virtualization under distinct approaches, and a resource-intensive application, like Hadoop needs that the performance overhead be reduced to be able to take advantage of virtualization systems.

According to our results, OpenVZ reach better rates and execution times than KVM on CPU intensive tests, I/O reading and small files I/O writing tests. KVM performed better results than OpenVZ only on the I/O writing tests of large files.

By using OpenVZ, a Hadoop cluster can achieve a high performance on virtualized systems when running jobs that use intensively CPU, network and I/O reading. Jobs that perform intense disk writing operations should be executed with caution or executed natively.

The overall effectiveness of OpenVZ allied to cloud computing features of OpenNebula brings OpenNebula OpenVZ suite as an effective Cloud Computing platform for Hadoop MapReduce clusters.

In the future, we plan to measure and analyze the performance of container-based and full virtualization platforms with open-source video-conference applications like Red5, BigBlueButton and others.

7. Acknowledgements

The authors would like to thank the Information Technology Center of the State University Vale do Acaraú (UVA) for allowing the use of infrastructure for development and testing of this work. Also, the authors thank to FUNCAP (Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico) for the financial support.

8. References

[1] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi. 2009. Evaluating MapReduce on Virtual Machines: The Hadoop Case., In Proceedings of the 1st International Conference on Cloud Computing (CloudCom '09), Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong (Eds.). Springer-Verlag, Berlin, Heidelberg, 519-528.

[2] J. Nilsson and P. Johansson, Hadoop MapReduce in Eucalyptus Private
Cloud., Bachelor's Thesis in Computing Science. Umea, Sweden, 2011.

[3] T. Mather and S. Kumaraswamy and S.Latif, Cloud Security and Privacy. O'Reilly, 2009.

[4] A. Devadiga, P.R Shalini and A. K. Sinha, Virtual Hadoop: The Study and Implementation of Hadoop in Virtual Environment using CloudStack KVM, International Journal of Engineering Development and Research (IJEDR), volume 2, issue 2, 2014.

[5] OpenNebula.org, OpenNebula — Flexible Enterprise Cloud Made Sim- ple, http://opennebula.org/, (Access Date: 04 Dec., 2014).

[6] J. Magnus and G. Opsahl, Open-source virtualization. Functionality and performance of Qemu/KVM, Xen, Libvirt and Virtualbox., Master's Thesis. Oslo, Norway, 2013.

[7] J. Che; Yong Yu; C. Shi; W. Lin, "A Synthetical Performance Evaluation of OpenVZ, Xen and KVM," *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific* , vol., no., pp.587,594, 6-10 Dec. 2010.

[8] Xavier, M.G.; Neves, M.V.; Rossi, F.D.; Ferreto, T.C.; Lange, T.; De Rose, C.A.F., "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments," Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on , vol., no., pp.233,240, Feb. 27 2013-March 1 2013 doi: 10.1109/PDP.2013.41

[9] Miguel G. Xavier and Marcelo V. Neves and Cesar A. F. De Rose,A Performance Comparison of Container-based Virtualization Systems for MapReduce Clusters, 22nd EuroMicro International Conference On Parallel, Distributed and network-based Processing (PDP2014), TuRin, Italy, IEEE, Feb 2014.

[10] J. P. Jacob and A. Basu, Performance Analysis of Hadoop Map Reduce on Eucalyptus Private Cloud,

International Journal of Computer Applications, volume 79, pags 10-13, 2013.

[11] T. White, Hadoop - The Definitive Guide, O'Reilly Media/Yahoo Press, 2nd edition, 2010.

[12] M. Zaharia and D. Borthakur and J. S. Sarma and K. Elmeleegy and S. Shenker and I. Stoica,Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling, 5th European conference on Computer systems (EuroSys '10), New York, USA, 265-278.

[13] A. Iordache, C. Morin, N. Parlavantzas, E. FelleR, P. Riteau, Resilin: Elastic MapReduce over Multiple Clouds Cluster, Cloud and Grid Com- puting (CCGrid), 13th IEEE/ACM International Symposium on Digital Object.

[14] Apache Hadoop, Hadoop 1.2.1 Documentation, http://hadoop.apache.org/docs/r1.2.1/hdfs design.html#Data+Replication, (Access Date: 04 Dec., 2014).

[15] Hui Kang, Yao Chen, Jennifer L. Wong, Radu Sion, and Jason Wu. 2011. Enhancement of Xen's scheduler for MapReduce workloads. In Proceedings of the 20th international symposium on High performance distributed computing (HPDC '11). ACM, New York, NY, USA, 251-262.

[16] C. He, D. Weitzel, D. Swanson, Y. Lu, HOG: Distributed Hadoop MapReduce on the Grid, SC Companion: High Performance Computing, Networking Storage and Analysis, 2012.

[17] G. N. Ramamoorthy. K-Means Clustering Using Hadoop MapReduce. Published by UCD School of Computer Science and Informatics.

[18] Kovari, A.; Dukan, P., "KVM & OpenVZ virtualization based IaaS open source cloud virtualization platforms: OpenNode, Proxmox VE," Intelligent Systems and Informatics (SISY), 2012 IEEE 10th Jubilee International Symposium on , vol., no., pp.335,339, 20-22 Sept. 2012

[19] Apache Hadoop, WordCount - Hadoop Wiki, http://wiki.apache.org/hadoop/WordCount, (Access Date: 04 Dec., 2014).

[20] O. O'Malley, TeraByte Sort on Apache Hadoop, Yahoo, 2008, http://sortbenchmark.org/YahooHadoop.pdf, (Access Date: 04 Dec., 2014).

