KVM, OpenVZ and Linux Containers: Performance Comparison of Virtualization for Web Conferencing Systems

Pedro Roger M. Vasconcelos, Gisele Azevedo A. Freitas, Thales G. Marques
Federal University of Ceara, Sobral, Ceara, Brazil

Abstract

Virtualization is an important technology in data center environment due to its useful features such as server consolidation, power saving, live migration and faster server provisioning. However, virtualization sometimes incurs some performance loss. Operating system-level virtualization could be an alternative to classical virtualization as it potentially reduces overhead and thus improves the utilization of data centers. Different virtualization platforms differ in terms of features, performance and virtualization overhead. Web conferencing systems become popular as the clients' bandwidth has increased in last years, in educational, researching and business fields. The BigBlueButton is a web conferencing system that allows multiple users join a conference room, having classes and share their microphone, webcam, desktop and files. In this paper, we use KVM, OpenVZ and Linux Containers as virtualization platforms to deploy conference systems using BigBlueButton. We explore its virtual performance under a real-world workload and a set of benchmarks that stress different aspects such as computing power, latency and memory, I/O and network bandwidth. These results can be a valuable information to be taken in account by systems administrators, for capacity planning and systems designing. Which, in turn, lead to cost savings for companies.

1. Introduction

The virtualization technologies allow partitioning the underlying hardware to multiple independent guest operating systems, coordinating and scheduling concurrent access to memory, CPU and devices. It has become increasingly popular in wide areas such as server consolidation, green IT and cloud computing. Advantages of consolidation include increased hardware resource use, reduced costs of servers, peripheral devices, cooling and power supply [1]. Flexible resource management, high reliability, performance isolation and operating system (OS) customization are other common features. The virtual machine monitor (VMM) is the piece in charge of providing the virtualization of underlying hardware and orchestration of concurrent resource dispute. By abstracting the underlying hardware into a generic virtual hardware, the VMM enables virtual machines (VM) to run in any available physical machines.

Several virtualization solutions have emerged, using different virtualization approaches. KVM and VMware use full virtualization to provide a complete virtual environment for a huge range of guest OSs. Xen can use paravirtualization to provide an API for guests domains, and solutions as OpenVZ and Linux Containers can provide ways to partition OS in logically isolated containers. Different virtualizations solutions have different strengths and weakness. For example, full virtualization supports both Linux and Windows virtual machines but the performance is relatively poor, while paravirtualization cannot support Windows virtual machine but the performance is better [2]. It is necessary to choose a most appropriate virtualization method for particular purposes. Besides these benefits, the VMM require additional CPU cycles and race conditions may occur causing overall performance degradation.

Multiple applications differ in the resource usage, which needs special attention in virtualized environments as bottlenecks can occur in the concurrent access to devices, or even the virtualization performance to the same kind of resource (e.g. CPU) can be different across different platforms. Based on the above analysis, it is essential to analyse the virtualization overhead, compare the efficiency of different VMMs and investigate bottlenecks that may occur in the usage of virtual resources. In this sense, the authors had evaluated the virtualization performance for other types of applications in previous works [3] and [4].

Video conferencing has been useful for years to the work meetings between offices of large companies and more recently in education, enhancing and easing the scope of teaching in schools and universities. The continuous increase in bandwidth of customers, companies and providers have contributed to the popularization of video conferencing solutions.

The BigBlueButton (BBB) platform is a popular web-based video conferencing system that makes use of open-source technologies to provide a complete environment to allow users create rooms, share presentations, audio and video.

Video conferencing systems can take advantage of several benefits of virtualization, they can be used together to achieve a best hardware usage. Server consolidation can optimize the use of the available hardware by making use of virtual machines to host video conferencing platforms on physical machines. Moreover, video conferencing systems can run for

long periods without use, which often cannot justify the adoption of a dedicated physical machine. Resources can be easily added to a virtualized video conferencing platform. And often, system administrators can make use of commodity server solutions instead of renting dedicated servers to host their video conferencing services, which ensures a huge economic advantage. So, evaluate and be aware of the performance of different types and virtualization platforms for such applications, become a valuable information for ensure quality of service and a significant contribution to enterprises from an economic point of view.

Thus, this paper has the aim to investigate the performance of BigBlueButton systems deployed over different MMVs under stress tests. The chosen virtualization platforms in this paper are KVM, OpenVZ and Linux Containers. Three open-source virtualization solutions that use two kind of virtualization approaches: full virtualization and OS-level virtualization. The authors have conducted several benchmarks from micro and macro perspective, evaluating the virtualization performance of specific resources and of the overall system.

The rest of this paper is structured as follows. In Section 2, we introduce a background of the VMMs used and an introduction to BigBlueButton. In Section 3 we present the related works. In Section 4 we present our experimental setup, macro and micro benchmarks, and an analysis of the performance achieved for both VMMs. Finally we give our conclusion in Section 5 and future work in Section 6.

2. Background

The term Virtual Machine took place in 1964 when IBM initiated a project called CP/CMS system what lead to the Virtual Machine Facility/370. In the 1970s, Goldberg and Popek wrote papers that gave a clear comprehension of the architecture [5] and requirements [6] related to virtualization.

The x86 OSs are designed to run directly on the bare-metal hardware, so they suppose they are the only operating system running on the underlying hardware. Commonly, the OS running in the physical machine is executed in kernel-mode and the user applications in user-mode. The x86 architecture has four levels of privileges known as Ring 0, 1, 2 and 3 used by OSs and applications to manage access to the computer hardware. The OS has direct access to memory and to the full set of CPU's instructions as it resides in the Ring 0. Applications running in the unprivileged user-mode, in Ring 3, have no direct access to the privileged instructions. The Ring 1 and 2 can be used for additional level of access, depending of the OS. Virtualization for the x86 architecture requires placing a virtualization layer under the OS, expected to be in the most privileged Ring 0, so that can be possible manage virtual machines and share resources.

VMs can be classified in System VMs, when are used to virtualize an entire OS, in opposition of Process VM such as those supported by the Java Virtual Machine [7].

There are two types of VMMs, type I and II [8]. A VMM type I, also called hypervisor or native, runs directly above the bare-metal hardware in the most privileged Ring, controls and shares resources to all VMs. The VMM type II runs as an application inside the OS, and is treated as a regular user space process.

In the next step, we describe the virtualization techniques to virtualize an entire OS. The current virtualization solutions can make use of many of virtualization techniques above to reach better performance. KVM and VMware make use of full virtualization, binary translation and hardware assist to virtualize almost any OS. These platforms provide paravirtualized drivers (VirtIO and VMware Tools, respectively) to allow their full virtualized guest OSs communicate in a more direct way with real hardware [9]. These drivers are not CPU paravirtualization solutions, they are minimal, non-intrusive changes installed into the guest OS that do not require OS kernel modification.

2.1. Full virtualization

Full virtualization can virtualize any x86 operating system using a combination of binary translation and CPU direct execution techniques. This approach translates kernel code to replace non-virtualizable instructions with new sequences of instructions that have the intended effect on the virtual hardware. Meanwhile, user level code is directly executed on the processor for high performance virtualization. The VMM provides each VM with all services of the physical system, including a virtual BIOS, virtual devices and virtualized memory management [9]. The guest OS is not aware it is being virtualized and requires no modification. Full virtualization is the only option that requires no hardware assist or operating system assist to virtualize sensitive and privileged instructions. The VMM translates all operating system instructions on the fly and caches the results for future use, while user level instructions run unmodified.

2.2. Binary translation

Binary translation is a technique used for the emulation of a processor architecture over another processor architecture. Thus, it allows executing unmodified guest OS by emulating one instruction set by another through translation of code. Un-like pure emulation, binary translation is used to trap and emulate (or translate) a small set of the processor

instructions[7]. That is, the code that needs privileged execution, in Ring 0, such as kernel-mode.

The rest of the instructions are executed directly by the host CPU.

2.3. Hardware assisted virtualization

Hardware assisted virtualization for the x86 architecture, was introduced in 2006 when Intel VT-x and AMD-V extensions were released. Hardware assisted virtualization implements a Ring with a higher privileged mode in the processor architecture.

These extensions support allows executing un-modified guest OSs in Ring 0 (non-root mode) and the hyper-visor in Ring -1 (root mode). Hardware assisted virtualization enhances CPUs to support virtualization without the need of binary translation or paravirtualization. The advantage is that this technique reduces the overhead caused by the trap-and-emulate model, instead of doing it in software here it is done in hardware.

2.4. Operating system-level virtualization

Here, the host OS is a modified kernel that allows the execution of multiple isolated containers (CT), also known as Virtual Private Server (VPS) or jail. Each CT is an instance that shares the same kernel of the host OS. Some examples that use this technique are Linux-VServer, OpenVZ and Linux Containers.

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 [7].

2.5. KVM

KVM (Kernel-based Virtual Machine) is a kernel-resident virtualization infrastructure for Linux on x86 hardware. KVM was the first hypervisor to become part of the native Linux kernel (2.6.20). KVM has support for symmetrical multiprocessing (SMP) hosts (and guests) and supports enterprise-level features such as live migration (to allow guest operating systems to migrate between physical servers) [10]. Because the standard Linux kernel is the hypervisor, it benefits from the changes to the mainline version of Linux (memory support, scheduler, and so on). Optimizations to these Linux components benefit both the hypervisor and the Linux guest OSs.

KVM is implemented as a kernel module, allowing Linux to become a hypervisor simply by loading a module. KVM provides full virtualization on hardware platforms that provide virtualization instructions support (Inter VT or AMD-V). KVM has two major components; the first is the KVM-loadable module that, when loaded in the Linux kernel, provides management of the virtualization hardware, exposing its capabilities through the /dev file system. The second component provides PC platform emulation, which is provided by a modified version of the QEMU emulator. QEMU executes as a user-space process, coordinating with the kernel for guest operating system requests.

When a guest OS is booted on KVM, it becomes a process of the host operating system and therefore scheduled like any other process. But unlike other process in Linux, the guest OS is identified by the hypervisor as being in the guest mode (independent of the kernel and user modes).

The KVM module exports a device called /dev/kvm which enables the guest mode of the kernel. With /dev/kvm, a VM has its own memory address space separate from that of the kernel or any other VM that is running. Devices in the device tree (/dev) are common to all user-space process, but /dev/kvm is different in that each process that opens it sees a different map, thus it supports isolation of the VMs [11]. Finally, I/O requests are virtualized through a lightly modified QEMU process that executes on the hypervisor, a copy of which executes with each guest OS process.

Other virtualization platforms have been competing to get into Linux kernel mainline for some time (such as UML and Xen), but because KVM required so few changes and was able to transform a standard kernel into a hypervisor, it's pretty clear why it was chosen.

2.6. OpenVZ

OpenVZ is an operating system-level virtualization techno-logy for the Linux kernel. It consists of a modified kernel that adds virtualization and isolation of various subsystems, resource management and checkpointing. OpenVZ allows a physical server to run multiple isolated operating system CTs.

Each CT is an isolated program execution environment that acts like a separate physical server. A CT has its own set of process starting from init, file system, users, networking interfaces with particular IP addresses, routing tables, etc. Multiple CTs can coexist on a single physical server, each one can operate different Linux distributions, but all CTs run under the same kernel [12], which results in excellent density, performance and manageability.

Virtualization and isolation enable many CTs within a single kernel. The resource management subsystem limits (and in some cases guarantees) resources, such as CPU, RAM, and disk space on a per-CTs basis. All those resources need to be controlled in a way that lets many CTs co-exist on a single system and not impact each other. The checkpointing feature allows stop a CT, saving its

complete state to a disk file, with the ability to restore that state later, even in another physical host.

The OpenVZ resource management subsystem consists of three components [13]:

- Two-level disk quota - Disk quotas can be set per-CT (first level) and inside CT, via standard Unix quota tools configured by the CT administrator.
- Fair CPU scheduler - The OpenVZ CPU scheduler is also two levels. On the first level it decides which CT to give the time slice to, taking into account the CT's CPU priority and limit settings. On the second level, the standard linux scheduler decides which process in the given CT to give the time slice to.
- User Beancounters - Is a set of per-CT counters, limits and guarantees. There is a set of about 20 parameters that cover all aspects of CT operation, so no single CT can abuse any resource that is limited for the whole computer and thus do harm to other CTs. The resources accounted and controlled are mainly memory and various in-kernel objects such as IPC shared memory segments, network buffers, etc.

While hypervisor-based virtualization provides abstraction for full guest OSs, operating system-level virtualization works at the operation system level, providing abstractions directly for the guest processes. OS-level virtualization shares the host OS and drivers with CTs and have smaller virtualization layer then hypervisors. Containers are more elastic than hypervisors and allow a higher density per node. Container slicing of the OS is ideally suited to cloud slicing, while the hypervisors' main advantage in IaaS is support for different OS families on one server.

2.7. Linux Containers

Linux Containers provides lightweight operating system-level virtualization and is relatively new to the other technologies, its 1.0 release was launched in 2014. Linux Containers source-code is included in the Linux mainline kernel.

Linux containers are a concept built on the kernel names-paces. This feature allows creating separate instances of previously-global namespaces. Linux implements file system, PID, network, user, IPC, and hostname namespaces. For example, each file system namespace has its own root directory and mount table, similar to chroot() but more powerful. Namespaces can be used in many different ways, but the most common approach is to create an isolated container that has no visibility or access to objects outside the container. Processes running inside the container appear to be running on a normal Linux system although they are sharing the underlying kernel with processes located in other namespaces [14].

The Linux control groups (cgroups) subsystem is used to group processes and manage their aggregate resource con-sumption. It is commonly used to limit the memory and CPU consumption of containers. A container can be resized by simply changing the limits of its corresponding cgroup. Cgroups also provide a reliable way of terminating all processes inside a container. Because a containerized Linux system only has one kernel and the kernel has full visibility into the containers there is only one level of resource allocation and scheduling [14].

An unsolved aspect of container resource management is the fact that processes running inside a container are not aware of their resource limits. For example, a process can see all the CPUs in the system even if it is only allowed to run on a subset of them; the same applies to memory. If an application attempts to automatically tune itself by allocating resources based on the total system resources available it may over-allocate when running in a resource-constrained container [14].

3. Related works

Xavier et al. [15] conducted experiments using the NAS Parallel Benchmarks (NPB) to evaluate the performance over-head and the Isolation Benchmark Suite (IBS) to evaluate the isolation in terms of performance and security. Since the focus is also on partitioning the resources of HPC clusters with multicore nodes, they evaluated the performance in both single and multinode environments, using respectively OpenMP and MPI implementation of NPB. It was evaluated Linux-VServer, OpenVZ, LXC and Xen. The worst result was observed in Xen for all I/O operations due to the paravirtualized drivers. These drivers are not able to achieve a high performance yet. In the network performance, 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 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. In the I/O performance, the worst result was observed in Xen for all I/O operations due to the paravirtualized drivers. These drivers are not able to achieve a high performance yet.

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. While LXC controls its resources only by cgroups, both Linux-VServer and OpenVZ implement their own capabilities introducing even more resource limits, such as the number of processes, which we have found to be an important contribution to give more security to the whole system. We suppose this capability will be introduced in cgroups in a near future. Careful examination of the isolation results reveals that all container-based systems are not mature yet. The only resource that could be successfully isolated was CPU. All three systems showed poor performance isolation for memory, disk and network. However, for HPC environments, which normally does not require the shared allocation of a cluster partition to multiple users, this type of virtualization can be very attractive due to the minimum performance overhead. Since the HPC applications were tested, thus far, LXC demonstrates to be the most suitable of the container-based systems for HPC. Despite LXC does not show the best performance of NPB in multinode evaluation, its performance issues are offset by the easy of 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.

Rizki et al. [16] conducted implementations of container-based virtualization, in this case OpenVZ and LXC. The experiments are benchmarking the virtualized clusters to measure the performances in terms of I/O performance overhead, and average time execution of mapreduce job. To measure the clusters performances, they used applications provided by hadoop packages. TestDFSIO used to measure the I/O performances of the cluster, WordCount used to measure the average time execution. TestDFSIO commonly used by hadoop users for identifying performances bottlenecks in networks, operating system, and also configurations of HDFS. MapRe-duce execution time are tested with syslog analysis to get and counts syslog priority in a dataset. Hosts kernel requirement are differ each other. In this experiments, OpenVZ stable used on top of Proxmox VE 3.4 which use kernel 2.6, while LXC used on top of Proxmox VE 4 which use kernel 3.1. Container-based hadoop cluster gives near native performances because of the lightweight virtualization on operating system level. In their experiment with 3 nodes on Proxmox VE using CFQ scheduler, the results shown that OpenVZ is more stable in current configurations. While in Syslog Priority Parser benchmarks, the results are comparative. But, OpenVZ gives more faster running time.

In the paper [17], authors aim to show the MapReduce (MR) as a platform for resource-intensive applications. So, container-based virtualization might be understood as a lightweight alternative to the traditional hypervisor-based virtualization systems. And, since there is a tendency to the usage of containers in MR clusters in order to supply resource sharing and performance isolation, experiments has been led to compare and contrast the usual container-based systems (Linux VServer, Open VZ and Linux Containers (LXC)) and MR clusters considering performance and manageability. The results indicate that all container-based systems reach a near-native performance for MapReduce workloads. Even so, LXC is the one which provides the best relationship between performance and management capabilities (notably concerning performance isolations).

4. Experimental methodology

4.1. Experimental setup

In order to evaluate the performance of BigBlueButton, we use the Proxmox Virtual Environment [18] to deploy two virtualized environments in a single machine. Proxmox is a complete open source server virtualization platform, that uses a custom optimized Linux kernel with KVM, Linux Containers (or depending of the version, OpenVZ) support. Proxmox offers enterprise-class features and a intuitive web interface to allow ease of management and deploying of virtual environments. The capability of use full virtualization to virtualize proprietary guest OSs (like Windows and macOS), and OS-level virtualization for easily deploy high density of containers in one single platform is a big advantage of Proxmox. Proxmox originally supported KVM and OpenVZ, until version 3.4. Since version 4.0 Proxmox adopted KVM and Linux Containers.

The physical machine is an IBM BladeCenter HS23 with two Intel Xeon CPUs E5-2620 of 2.00 GHz (with 6 cores each and Hyper-Threading technology), 48 GB of RAM, connected to a local gigabit ethernet network, in turn, connected to the Internet through a 100 Mbps dedicated link. Because Proxmox only support OpenVZ until version 3.4, we had to install two Proxmox environments: Proxmox VE 3.4 kernel 2.6.32-37-pve amd64 and Proxmox VE 4.3-1 kernel 4.4.19-1-pve amd64.

It was deployed one OpenVZ, one Linux Containers CT and one KVM VM, both using Ubuntu GNU/Linux 10.04.4 LTS AMD64. Each virtual environment runs BigBlueButton version 0.81 and has 2 virtual vCPUs, 4 GB of RAM, 15 GB storage size, and one virtual gigabit ethernet adapter.

We use a set of two physical machines and 120 Amazon EC2 and Google Compute Engine instances (a custom set of t2.micro and g1-small instances type running Windows Server 2012 R2) to simulate conference clients in order to stress the virtualized

BBB servers. We utilize the v4l2loopback software to simulate a webcam device in each workstation and use Mplayer to send a video file to those devices, thus we could simulate a complete video conference client.

We use the Zabbix network monitoring software in order to collect data from the virtualized servers. Those data include the usage of CPU, memory, I/O, ingress/egress network traffic and number of conference sessions.

4.2. Macro benchmarks

In this section we observe the performance and resource utilization of BBB while users are continuously joining the conference. We begin the benchmark without users connected into BBB and gradually one user joins the videoconference every 30 seconds. The two workstations in the local network open 4 conference clients each and begin streaming their virtual webcams. The first connected user also share his own desktop screen through the BBB's java applet. We use Mozilla Firefox browser version 46 to open 4 BBB sessions in each workstation. BBB treats each tab in the browser as an independent video conference client.

In the next step, we begin launching the Amazon EC2 and Google Compute Engine instances. Each instance opens 4 sessions to BBB servers through the Internet. Instances do not own webcams, thus they just watch the video from other users. The benchmark stops when it reaches 340 connected users. A benchmark run with 340 simultaneous users is shown in Fig. 1.

Figure 1. BBB session with 340 users

The Fig. 2 shows the CPU utilization according the number of connected users. As explained above, inside a LXC container, processes see the entire host hardware, e.g. all CPUs. So we could not use the Zabbix agent to collect CPU usage statistics because it sees the 24 CPU cores instead of the 2 cores allocated to the CT. This is a drawback of Linux Containers, we must have user space programs that be cgroup-aware to collect performance data. Recently it was released the project LXCFS, that allows bind mount a /proc file system cgroup-aware in each container, but it was not compatible with the Ubuntu version used. So, for Linux Containers the CPU usage was collected through Proxmox API.

The CPU usage for Linux Containers is very above the values of KVM and OpenVZ, this can be correlated with the way the CPU usage is gathered (Proxmox API). We can observe that the BBB hosted into KVM virtual machine presents a slightly increase in CPU usage. KVM presented CPU usage difference peak 19.2 % higher than OpenVZ. However, the CPU usage average of the entire benchmark for KVM is only 8 % higher of OpenVZ CPU use.

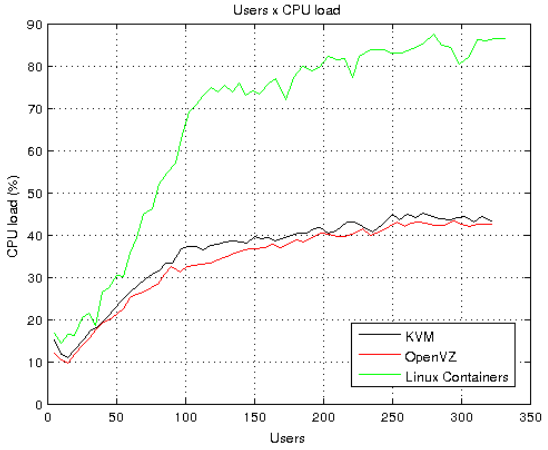

Figure 2. CPU usage by connected users

Fig. 3 illustrates the memory consumption by the number of connected users into the BBB session. Here we can observe that the VM presents a memory consumption 11.6% higher compared to OpenVZ, and Linux Containers presented 10.27% higher than OpenVZ. This can be explained mainly by the minor footprint of OpenVZ containers (fewer process, modules, daemons and virtualization layers compared to a full Linux system in KVM).

4.3. Micro benchmarks

In order to investigate the performance data shown above, we ran a collection of micro benchmarks to evaluate the virtualization performance of individual resources. Thus, we can analyse if the performance in some macro

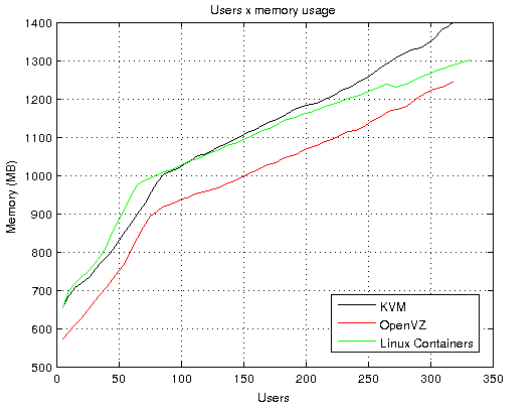

Figure 3. Memory usage by connected users

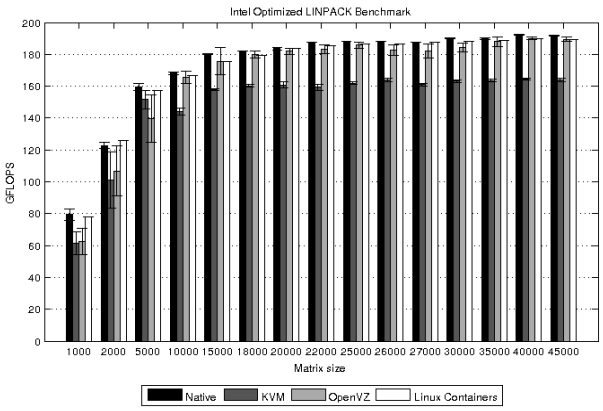

Figure 4. LINPACK benchmark

benchmark is result of poor performance of a individual, or group of, resources.

CPU virtualization: We evaluate the CPU virtualization performance of KVM and OpenVZ with the Intel Optimized LINPACK Benchmark (version 11.3.3.011). It consists of a generalization of the LINPACK 1000 benchmark. It solves a dense system of linear equations, measures the amount of time it takes to factor and solve the system, converts that time into a performance rate, and tests the results for accuracy. The generalization is in the number of equations it can solve, which is not limited to 1000. It uses partial pivoting to assure the accuracy of the results [19].

The Intel LINPACK Benchmark is threaded to effectively use multiple processors. So, in multi-processor systems, best performance will be obtained with the Intel Hyper-Threading Technology turned off, which ensures that the operating sys-tem assigns threads to physical processors only. Intel LIN-PACK is based on the Intel Math Kernel Library, which is highly adaptive and optimizes itself based on both available floating point resources, as well the cache topology of the system. Thus, to achieve maximum performance we activate the exposing of host system's NUMA topology to KVM VMs in Proxmox web interface.

We run LINPACK over 12 CPU cores on the host ma-chine, VM and CTs. We use matrices with orders ranging from 1000 until 45000. The Fig. 4 shows the results of LINKPACK benchmark. Performance is almost identical for most of the problems sizes between the host machine and containers. The average GFLOPS values for KVM, OpenVZ and Linux Containers are about 86.05%, 95.42% and 99.13% to native system, respectively. The vast majority of operations in LINPACK are spent in double-precision floating point operations, the gathered data reveals that efficiency of KVM on floating point computing is not so good as OpenVZ and Linux Containers.

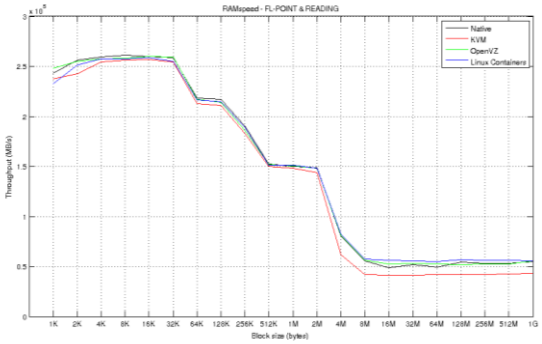

Figure 5. RAMspeed benchmark for 8 threads

Memory virtualization: To investigate the memory virtualization performance we choose RAMspeed (version 3.5.0). RAMspeed is an open-source utility to measure cache and memory performance. It has numerous benchmarks such as INTmark, FLOTmark, MMXmark and SSEmark. They operate with linear (sequential) data stream passed through ALU, FPU, MMX and SSE units respectively. They allocate certain mem-ory space and start either writing to, or reading from it using continuous blocks sized of 1 MB. This simple algorithm allows to show how fast are both cache and memory subsystems. It was used 8 GB per pass and 8 CPUs in RAMspeed runs.

The Fig. 5 illustrates the performance of memory virtualization varying the block size in float-pointing reading RAM-speed test. We can observe the effect of the dual Xeon E5-2620's three levels of on-chip cache, observed by throughput drops near 32 KB, 256 KB and 2 MB. The L1 (32KB data) and L2 (256KB) caches are dedicated to each core. The L3 cache (15 MB) is shared over the 6 cores in a single CPU, what theoretically can reach 2.5 MB per core. The performance become fairly stable for data blocks after 16 MB, when the effects of all cache are overwhelmed. This space represents the speed of the memory subsystem, and ensures no cache was involved [20]. All three systems reach near same

throughput while the operations involves the cache system, while for bigger block sizes, when the cache effects are minimized, KVM reaches 89.9% of native speed and the containers follow native performance.

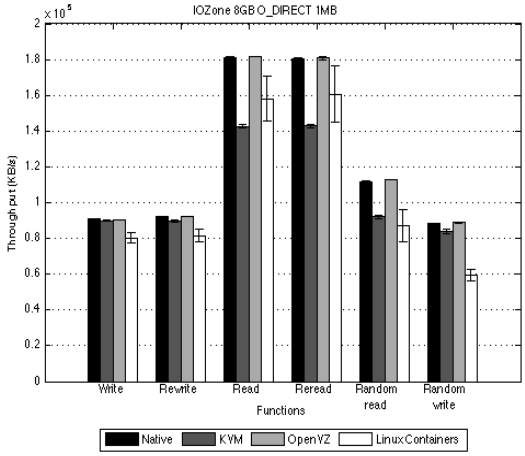

Figure 6. IOzone throughput

Disk virtualization: Following, we analyse how fast the execution environments can perform disk operations. We choose the IOzone tool to assess the disk I/O performance. IOzone generates and measures a variety of file operations. In the native system, we create an ext4 file system in the local RAID 5 array using the mkfs default settings and mount it normally. In the OpenVZ container we mount it using the OpenVZ's simfs file system. Simfs is not an actual file system, it is a map to a directory on the host, like a proxy file system. This file system allows isolating a particular container from other containers. In the VM we choose a LVM logical volume created in the same RAID 5 array, and mount it as a virtIO block device. Inside the VM the device was formatted as a default ext4 volume. We made IOzone runs of 8 GB total size and 1024 KB record size in O_DIRECT mode.

Using O_DIRECT allows access to bypass the buffer cache and go directly to disk. Because Linux kernel makes use of I/O cache in memory, we also reduce the available RAM in all environments to 4GB. Linux Containers, however, apparently does not support the O DIRECT flag, thus all the I/O requests are cached in the host memory, even the container with 4GB of RAM, in the benchmark we see more than 8GB of cached memory on the host system. Thus, we could not compare cached I/O access with no cached ones. As workaround, we tried to drop the cache at the host system with the command "echo 3 > /proc/sys/vm/drop caches" while benchmarking Linux Containers.

Fig. 6 shows almost no losses in throughput for write and rewrite operations for KVM and OpenVZ. Although, while KVM reaches 78.9%, 79.3%, 82.6% and 95.2% for read, reread, random read and random write,- respectively, Linux Containers reaches 87.45%, 89.26%, 78.11% and 67.40%. OpenVZ reaches near 100% in relation to native in the same operations. This performance gap in KVM maybe can be explained by the fact that each I/O operation must go through QEMU in user space.

Network virtualization: The network performance is evaluated with the Netperf benchmark. Netperf is a tool for measurement of network throughput and end-to-end latency. In our setup another blade server run the netperf server, called netserver, and the benchmark's target machine runs the netperf client. As shown in Table 1, all environments reach the same performance, near the gigabit ethernet interface limit, revealing no losses in network bandwidth between KVM, OpenVZ and Linux Containers, compared to native Linux.

Table 2 reveals the network latency for TCP and UDP communications. Netperf does a request/response for a given request and response size, and measures the round-trip average latency.

The round-trip latency for TCP is naturally bigger due the inner overhead of TCP 3-way handshake and flow control. In OpenVZ CTs the network is provided through the venet interface (loaded by the vznetdev kernel module), a virtual interface that acts like a point-to-point connection between the container and the host system. In KVM, we use the paravirtualized virtIO network interface. This guest device is connected in a called tap device on the host, which in turn, joins a bridge with the host's physical interface. Linux Containers use the same approach that KVM to virtualize network. Both virtualization approaches add some latency overhead. Although, for TCP, KVM adds 91.7% of latency, while OpenVZ increases it by 22% and Linux Containers adds 67.4%. For the UDP protocol, KVM adds a latency 119.5% higher than the physical host while OpenVZ adds 11.9% and Linux Containers 82.87%. The increased number of layers in the virtual network device can contribute to the higher latencies for KVM and Linux Containers.

Table 1. Netperf bandwidth

	Native	KVM	OpenVZ	Linux Containers
TCP	935	933	935	935
UDP	954	952	954	954

Table 2. Netperf round-trip time latency

	Native	KVM	OpenVZ	Linux Containers
TCP_RR	77.5	148.6	94.56	129.7
UDP_RR	64.95	142.5	72.7	118.7

5. Conclusion

As we have shown, video conference systems are a kind of application well suited to virtualization. Performance evaluation of virtualization platforms is an important key in improving their design and implementation. We assess KVM, OpenVZ and Linux Containers through a real-world application, the BigBlueButton conference system, and a series of micro benchmarks. The results show that OpenVZ presents a better overall performance followed by Linux Containers. KVM presents slight performance degradation in the most cases. The performance overhead of CPU and memory for KVM presented in the macro benchmarks, is confirmed in the micro benchmarks. OpenVZ shows a clear performance gain in some I/O operations such as storage read, reread, random read, and in memory access bandwidth, as well. The higher round-trip network latency presented by KVM and Linux Containers should be taken into account when designing services latency-sensitive, like video conference and real-time systems.

This increased latency joint with high load of CPU and network, due to a large number of sessions, can be determinant to a poor quality of these services. The fact that Linux Containers does not abstract the hardware seen by the CT can be a problem to applications that make use of hardware detection to do auto adjust. OpenVZ and KVM guests are much more clear in this sense and this could have influenced in the benchmarks results. Thus, OpenVZ demonstrates to be most suitable virtualization platform for video conferencing workloads, also presenting minor container footprint, what leads to a higher container density, allowing more containers running on the same host. On the other hand, KVM presents a more convenient way for guest-independent virtualization. Thus, the results can characterize a trade-off between performance gain and ease of use.

6. Future work

As future work it is interesting investigate and evaluate the performance of Linux Containers with LXCFS. LXCFS is a set of files which can be bind-mounted over their /proc originals to provide cgroup-aware values. Thus performance measurement tools can take a clear comprehension of the hardware.

7. References

[1] H. Oi and K. Takahashi, “Performance modeling of a consolidated java application server,” in 2011 IEEE International Conference on High Performance Computing and Communications, Sept 2011, pp. 834–838.

[2] K. Ye, J. Che, Q. He, D. Huang, and X. Jiang, “Performance combinative evaluation from single virtual machine to multiple virtual machine systems,” International Journal of Numerical Analysis and Modeling, vol. 9, no. 2, pp. 351–370, 2012.

[3] P. R. M. Vasconcelos and G. A. de Arajo Freitas, “Performance analysis of hadoop mapreduce on an opennebula cloud with kvm and openvz virtualizations,” in The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014), Dec 2014, pp. 471–476.

[4] P. R. M. Vasconcelos and G. A. de Araujo Freitas, “Evaluating virtual-ization for hadoop mapreduce on an opennebula clouds,” International Journal Multimedia and Image Processing, vol. 4, pp. 234–244, Dec. 2014.

[5] R. P. Goldberg, “Architecture of virtual machines,” in Proceedings of the Workshop on Virtual Computer Systems. New York, NY, USA: ACM, 1973, pp. 74–112.

[6] G. J. Popek and R. P. Goldberg, “Formal requirements for virtualizable third generation architectures,” Commun. ACM, vol. 17, no. 7, pp. 412– 421, Jul. 1974.

[7] F. Rodrguez-Haro, F. Freitag, L. Navarro, E. Hernnchez-snchez, N. Faras-Mendoza, J. A. Guerrero-Ibez, and A. Gonzlez-Potes, “A summary of virtualization techniques,” Procedia Technology, vol. 3, pp. 267 – 272, 2012.

[8] R. P. Goldberg, “Architectural principles for virtual computer systems,” DTIC Document, Tech. Rep., 1973.

[9] VMware, “Understanding full virtualization, paravirtualization, and hardware assist,” VMware Inc., Tech. Rep., Mar. 2008.

[10] M. T. Jones, “Anatomy of a linux hypervisor,” IBM, Tech. Rep., May 2009.

[11] “Discover the linux kernel virtual machine,” IBM, Tech. Rep., May 2007.

[12] K. Kolyshkin. Virtualization comes in more than one flavor. [Online]. Available: http://kirillkolyshkin.ulitzer.com/node/318829 (Access Date: 12 May, 2016)

[13]Openvz virtuozzo containers wiki. [Online]. Available: http://www. linux-kvm.org/page/Main Page (Access Date: 12 May, 2016)

[14]W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated perfor-mance comparison of virtual machines and linux containers,” in 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), March 2015, pp. 171–172.

[15] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose, “Performance evaluation of container-based virtual-ization for high performance computing environments,” in Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, ser.

PDP ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 233–240.

[16] R. Rizki, A. Rakhmatsyah, and M. A. Nugroho, “Performance analysis of container-based hadoop cluster: Openvz and lxc,” in 2016 4th In-ternational Conference on Information and Communication Technology (ICoICT), May 2016, pp. 1–4.

[17] M. G. Xavier, M. V. Neves, and C. A. F. D. Rose, “A performance comparison of container-based virtualization systems for mapreduce clusters,” in 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Feb 2014, pp. 299–306.

[18] Proxmox. Proxmox - powerful open source server solutions. [Online]. Available:http://www.proxmox.com/ en/ (Access Date: 23 July, 2016).

[19] Intel. Intel(r) optimized linpack benchmark for linux* os. [Online]. Available: https://software.intel.com/en-us/node/528615 (Access Date: 23 July, 2016)

[20] S. C. C. Bell and R. Radcliff, “Designing a memory benchmark,” Deopli Corporation, Tech. Rep., 2011.

8. Acknowledgements

The authors would like to thank the Information Technology Center (NTI) of the State University Vale do Acaraú (UVA), for the support and for allowing the use of its computing infrastructure for the deployment, development and testing of the presented research.

