Journal of Theoretical and Applied Information Technology
28th February 2014. Vol. 60 No.3
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A PERFORMANCE STUDY OF HARDWARE IMPACT ON FULL VIRTUALIZATION FOR SERVER CONSOLIDATION IN CLOUD ENVIRONMENT

[1]S.SURESH, [2]M.KANNAN
[1]Associate Professor, Department of CSE, Adhiyamaan College of Engineering, Hosur-635109,Tamil Nadu
[2]Professor, Department of IT, Sri Ramakrishna Institute of Technology, Coimbatore-641 010,India
E-mail: [1]ssuresh.siv.72@gmail.com, [2]kannankrish68@yahoo.com

ABSTRACT

Underutilization of hardware resources has always been a problem in single workload driven traditional OS environment. To improve resource utilization, virtualization of multiple VMs and workloads onto the same host with the aid of Hypervisor has been the recent trend. Its use cases such as server consolidation, live migration, performance isolation and on-demand server provisioning make it as a heart part of enterprise application cloud. Cloud is an on-demand, service provisioning technology, where performance plays a vital role for user acceptance. There are numerous virtualization technologies are available from full virtualization to paravirtualization, each has its strength and weakness. As performance study is an ongoing pursuit, hardware and software development getting matured day by day, it is desirable to do this sort of performance study in regular interval that often sheds new light on aspects of a work not fully explored in the previous publication. Hence, this paper focus performance behaviours of various full virtualization models such as hosted (VirtualBox) and bare metal (KVM) virtualization using variety of benchmarks from micro, macro and application level in the cloud environment. We compare both virtualization solutions with a base system in terms of application performance, resource consumption, low-level system metrics like context switch, process creation, interprocess communication latency and virtualization-specific metrics like virtualization layer consumption. Experimental results yield that VirtualBox outperforms KVM in CPU and thread level parallelism and KVM outperforms in all other cases. Both are very reluctantly accepted for disk usages comparing with native system.

Keywords: *Full Virtualization, VirtualBox, KVM, Server Consolidation, Performance*

1. INTRODUCTION

Virtualization provides an abstract way for different servers to co-exist on the same server machine and share resources, while the underlying virtualization layer offers isolation and performance guarantees. The building block of this abstraction is the virtual machine (VM), which can accommodate the whole server application or parts of it. Multiple different server applications, even running on heterogeneous operating systems can be hosted by the same physical machine. Hence server consolidation is defined as the process of encapsulating single server workload into VMs and running them in a shared hardware platform via, virtual machine monitor (VMM) or hypervisor. That simplifies load balancing, dealing with hardware failures and eases system scaling. In addition to share resources promises a more efficient usage of available hardware.

Virtualization is the engine that drives cloud computing [1],[2] by turning the data center into self-managing, highly scalable, highly available, pool of easily consumable resources. Virtualization and, by extension, cloud computing provide greater automation opportunities that reduce administrative costs and increase the company's ability to dynamically deploy solutions. By being able to abstract the physical layer away from the actual hardware, cloud computing creates the concept of a virtual data center, a construct that contains everything in physical data center. While there are currently a number of virtualization technologies available today, the virtualization technique of choice for most open platforms over the past years have typically been the Xen hypervisor because of its performance. Though paravirtualization performance was slightly better than full virtualization, full virtualizations strength on paravirtualization is superior security, it's cleanliness diagram, heterogeneity OS support and hardware advancement get the attention of the enterprise to switch to the full virtualization.

The combination of new CPU architectures with embedded virtualization support and advances in hypervisor design have eliminated many of their performance overheads. Despite this, popular hypervisors still exhibit different levels of performance. To understand the relative strengths and weaknesses of different hypervisors, in this paper we perform an extensive performance comparison of two open source full virtualization platforms VirtualBox and KVM. Hence, this sort of performance study often sheds new light on aspects of a work, not fully explored in the previous publications[3].The objective of this research work is to figure out the following questions: a)The performance degradation of virtual machine against physical machine? b) How much difference between different virtualization technologies? c) What factors lead to the performance loss of virtualization systems? We limit our performance analysis to open-source full virtualization solutions with hardware support, as their user license allows us to publish the results without any restriction. The focus of our analysis is on the virtualization of 64-bit guests over 64-bit hosts. Our study was motivated by the interests in using virtualization technology in both single and multiple virtual machine system.

Our paper specifically make the following contributions: a) we've described tools to measure performance, ranging from the general to the specific, and from the hardware focused to the application oriented b) we present a detailed measurement based performance characterization of the typical server virtualization workloads. c) We show VM workload interactions and interference under different degrees of resource sharing. d) Based on the earlier observations, in this paper, we evaluate two representative full virtualization technologies, VirtualBox and KVM, in various configurations. We consolidate three more VM systems to drive the system with and without workload. We compare both technologies with a base system in terms of application performance, resource consumption, low-level system metrics like context switch, inter process communication latencies and virtualization-specific metrics like virtualization consumption.

We discuss the related research efforts in Section 2. We provide background of full virtualization technologies and hardware enhancements in Section 3. Section 4 presents our evaluation approach and findings of a measurement-based study are reported in Section 5. We conclude with a discussion of findings and future directions of this work in Section 6.

2. RELATED WORK

There are lot of works are carried out in performance study in all aspects, ranging from performance study of single hypervisor or multiple hypervisor and reducing the overhead incurred by the virtualization layer and characterize and analyze the performance impact of various types of workloads on VMs.

In [4] authors have compared performance of software VMM with new designed VMM that employs recent hardware extensions support. The experiments result different from the perception, hardware VMM fails to provide a certain performance enhancement. The reasons cause this situation is analyzed in the paper. Barham et al. [5] present a comprehensive introduction to the Xen hypervisor and compare its performance to a native system, the VMware workstation and a User-Mode Linux at a high level of abstraction. They show that the performance is practically equivalent to a native Linux system and state that the Xen hypervisor is very scalable. Deshane [6] presented an independent research describing the performance comparison between Xen and KVM, which evaluated the overall performance, security impacts, performance isolation and scalability of Xen and KVM. In [7], Menon et al. present a diagnosing performance overhead method about resource scheduling in the Xen virtual machine environment. In this method, a toolkit is used to analyze performance overheads incurred by networking applications running in Xen VMs. The toolkit enables coordinated profiling of multiple VMs in a system to obtain the distribution of hardware events such as clock cycles and cache and TLB misses. In [8], Ye et al. provide a framework to analyze the performance of virtual machines system, which is based on the queuing network models. In the framework, the virtual machines either do not run at all or just monitor the virtual machines instead of the hypervisor. Apparao et al. [9] analyze the performance characteristic of a server consolidation workload. Their results show that most of the performance loss of CPU intensive workloads is caused by cache and core interferences. However, since the publication of these results, the considered virtualization platforms have changed a lot (e.g., hardware support was introduced) which renders the results outdated. Hence, the results of these works must be revised especially to evaluate the influences of, e.g., hardware support.Tickoo et al. [10] identifies the challenges of modeling the contention of the visible and invisible resources and the hypervisor. In their consecutive work based on measure and model the influences of VM shared resources. They show the importance of shared resource contention on virtual machine performance and model cache and core effects, but no other performance-influencing factors. In [11], authors have mentioned their clear procedure and result of benchmarking to analyze performance of openVZ, Xen and KVM hypervisors. In their research, a combine result of processor performance, network performance, database server performace, disk performance has reflected the overall system performance. Low-level benchmarking like context switching has revealed the micro performance of system. Authors [12] have described benchmarking tools; Linpack, Lmbench and IOzone in their paper. They have provided a series of performance experiment during the testing of Xen and KVM hyprevisors. By doing CPU overhead analysis, memory bandwidth analysis, I/O operating analysis, they have tried to figure out main source of the total virtualization overhead. In this paper [13], author use the automated experimental analysis approach to evaluate its applicability to Citrix XenServer and VMware ESX. His aim is to build a generic model which enables the prediction of performance overheads on different virtualization platforms. In addition, they evaluate various performance influencing factors like scheduling parameters, different workload types and their mutual influences, and scalability and over commitment scenarios using passsmark and Iperf.In [14] author quantify, model and predict the virtualization performance overhead by capturing the performance relevant factors explicitly and focus on specific aspects using queueing network models. They claimed significant improvements in the prediction accuracy in their approach by evaluating for various scenarios based on the SPECjEnterprise2010 standard benchmark and XenServer. In [15] author has performed an extensive performance comparison using under hardware-assisted virtualization settings for different virtualization solutions, Hyper-V, KVM, vSphere and Xen. They use a component-based approach that isolates performance by resource such as CPU, memory, disk, and network. Further they study the level of performance isolation provided by each hypervisor to measure how competing VMs may interfere with each other and find that the overheads incurred by each hypervisor can vary significantly depending on the type of application and the resources assigned to it.

Still, none of the researchers are aware to address some major issues on virtualization performance collectively, such as consolidation issue (with and without workload), benchmark issue (ranging from system level, component level, application level from simple to multi threaded) and interference issue, etc., with the recent hardware and software advancements. Further, one can see a lot of papers on KVM and a few on VirtualBox virtualization solutions.

3. OVERVIEW OF FULL VIRTUALIZATION TECHNOLOGIES

Hardware virtualization means abstracting functionality from physical components. This principle is used for sharing hardware by multiple logical instances with the help of the virtual machine monitor / hypervisor. The hypervisor gives each guest domain a portion of the full physical machine resources. Multiple guests running on the same physical machine must share the available resources. Therefore, the hypervisor does not generally expose the full power of the underlying machine to any one guest. Instead, it allocates a portion of the resources to each guest domain. It can either attempt to partition resources evenly or in a biased fashion to favour some guests over others. It grants each guest a limited amount of memory and allows each guest only its fair share of the CPU. Similarly, it may not want all guests to have access to every physical device in the system and thus it only exposes the devices it wants each guest to see. Sometimes, it may even create virtual devices that have no corresponding underlying physical device—for example, a virtual network interface [16]. There are two forms of full virtualization available.

3.1 Type 1 or Bare-metal Architecture

In this architecture, virtual machine monitor is responsible for controlling the operating system environment by scheduling and allocating resources to all virtual machines running in the system. It is believed that this hypervisor delivers high performance and better scalability.

3.1.1 KVM

KVM (Kernel-based Virtual Machine) [17] is an open-source hypervisor using full virtualization capable of running heterogeneous VMs. As a kernel driver added into Linux, KVM enjoys all advantages of the standard Linux kernel and hardware-assisted virtualization KVM introduces virtualization capability by augmenting the traditional kernel and user modes of Linux with a new process mode named guest, which has its own kernel and user modes and answers for code execution of guest operating systems.KVM comprises two components: one is the kernel module and another one is userspace. Kernel module (namely kvm.ko) is a device driver that presents the ability to manage virtual hardware and see the virtualization of memory through a character device /dev/kvm. With /dev/kvm, every virtual machine can have its own address space allocated by the Linux scheduler when being instantiated. The memory mapped for a virtual machine is actually virtual memory mapped into the corresponding process. Translation of memory address from guest to host is supported by a set of page tables.KVM can easily manage guest Operating systems with kill command and /dev/kvm.User-space takes charge of I/O operation‘s of virtualization. Further, it also provides a mechanism for user-space to inject interrupts into guest operating systems. User-space is a lightly modified QEMU, which exposes a platform virtualization solution to an entire PC environment including disks, graphic adapters and network devices. Any I/O requests of guest operating systems are intercepted and routed into user mode to be emulated by QEMU.

3.2 Type 2 or Hosted Architecture

The principle of this architecture is that virtual machine monitor runs on extended host under the host operating system, which means that hypervisor runs as an application on the host operating system and guest operating system runs inside hypervisor. In this approach, hypervisor has higher privilege level than guest OS kernel, and it separates the operating system from the hardware resource logically. In full virtualization technology, guest OS kernel does not need to modify, hence it not only reduces managing efforts but also allows creating virtual environment using operating systems of close-source type. Unlike paravirtualization, it facilitates modularization by separating hardware and/or software into functional components thus customization of VM’s setup is possible. Full-virtualization also offers secured migration and delivers perfect isolation of guest OS from the underlying hardware so that its instance can run on both virtualized and non-virtualized conditions. Unfortunately, its layered architecture possesses security management complexity and incurs performance penalty. Furthermore, it delivers high system overhead, as it is responsible for caring all the system activity through the hypervisor.

3.2.1 VirtualBox

Oracle VM VirtualBox [18] is an x86 cross platform open source virtualization software package. VirtualBox is a so-called "hosted" hypervisor, a software technology that can run without the hardware help but capable of using hardware extensions. The CPU extensions let it to run its guest OS, running in ring 1, is reconfigured to execute in ring 0 on the host hardware when OS request trap to VMM. As some code contains privileged instruction must be intercepted and rewritten to manipulate protected resources which cannot run natively in ring 1, VirtualBox employs a Code Scanning and Analysis Manager (CSAM) to scan the ring 0 code recursively before its first execution to identify problematic instructions and then calls the Patch Manager (PATM) to perform in-situ patching. This replaces the instruction with a jump to a VM-safe equivalent compiled code fragment in hypervisor memory. The guest user-mode code, running in the ring 3, is generally run directly on the host hardware at ring 3. In hardware assisted emulation, it provides the option to enable hardware virtualization on a per virtual machine basis. On more recent CPU designs, VirtualBox is also able to make use of nesting paging tables, which can greatly accelerate hardware virtualization since these tasks no longer need to be performed by the virtualization software.

3.3 Hardware virtualization

Recognizing the importance of virtualization, hardware vendors Intel[19],AMD [20] have added extensions to the x86 architecture that make virtualization much easier. Intel’s *Virtualization Technology for x86* (*VT-x*) and AMD’s *Secure Virtual Machine* (*SVM*). Both provide a higher privilege mode than ring 0, in which a hypervisor can sit without having to evict the kernel from ring 0. This separation is particularly important on x86-64, because it means that the kernel does not have to run at the same privilege level as the applications, and so no tricks are required to allow it to poke around in their address spaces. These new processors allow trapping of sensitive events. This eliminates the need for binary translation and simplifies the hypervisor. The biggest difference between Intel’s VT-x and AMD’s SVM comes is, AMD moved the memory controller on-die, whereas Intel kept theirs in a discrete part. Hence, AMD was able to add some more advanced modes for handling memory. With VT-x, one simply set a flag that causes page table modifications to be trapped. SVM provides two hardware-assisted modes. The first, *Shadow Page Tables* allows the hypervisor to trap whenever the guest OS attempts to modify its page tables and change the mapping itself. This is done, in simple terms, by marking the page tables as read only, and catching the resulting fault to the hypervisor, instead of the guest operating system kernel. The second mode is a little more complicated. *Nested Page Tables* allow a lot of this to be done in hardware. Nested page tables do exactly what their name implies; they add another layer of indirection to virtual memory. The MMU already handles virtual to physical translations as defined by the OS. Now, these "physical" addresses are translated to real physical addresses using another set of page tables defined by the hypervisor. Because the translation is done in hardware, it is almost as fast as normal virtual memory lookups. Memory handling and switching is also boosted by tagged Translation Lookaside Buffer (TLB) capabilities that map memory space to the individual VM. This reduces memory management and speeds up the switching process between VMs. The other additional feature of hardware extension to devices is that it specifies a *Device Exclusion Vector* interface. This masks the addresses that a device is allowed to write to, so a device can only write to a specific guest's address space. Intel also introduced VT-d, which is Intel's technology for Direct IO. These extensions allow for devices to be assigned to virtual machines safely. VT-d also handles Direct Memory Access (DMA) remapping and the I/O translation look aside buffer (IOTLB). DMA remapping prevents a direct memory access from escaping the boundaries of the VM. IOTLB is a cache that improves performance. By comparison, the VT-d extensions add virtualization support to Intel chipsets that can assign specific I/O devices to specific VMs, while the VT -c extensions bring better virtualization support to I/O devices such as network switches.AMD also introduced a new technology to control access to I/O called I/O Memory Management Unit (IOMMU), which is analogous to Intel's VT-d technology. IOMMU is in charge of virtual machine I/O, including limiting DMA access to what is valid for the virtual machine, directly assigning real devices to VMs.

4. EXPERIMENTAL METHODOLOGY

4.1 Experimental Setup

All the experiments were conducted on physical hardware configured with ASUSTek computer Inc motherboard AMD64 780G chipset model M5A78L-M Lx V2 AMD Fx-8150 Eight-core desktop processor ,8 GB DDR3 RAM,L2 cache 2048 Kbytes,L3 8Mbytes ,100Mbits network card. Both host and virtual machine is configured with 8 VCPUs and 2 GB RAM, 60GB HDD with Ubuntu 13.10 (Saucy Salamander). The virtualization solutions considered are KVM 76, VirtualBox 4.2. In all solutions we use hardware virtualization support to virtualize 64-bit guests over a 64-bit host. For the KVM and VirtualBox machines, virtual NICs are using the default bridged network driver. The cloud environment is emulated using Dummynet [21] and the experimental setup mimic as shown in the figure 1. *Performance Baseline:* For establishing a performance baseline, we use Ubuntu 13.10 Linux kernel without virtualization to run all benchmarks with one to eight threads to measure the scalability with respect to the number of threads. For baseline network I/O measurement, the client and server threads running on different hosts connected through (emulated) WAN is taken. The first sets of results represent the performance of various benchmarks. Each benchmark was run a total of 20 times, and the mean values taken with error bars represented using the standard deviation over the 20 runs.

4.2 Benchmarks

We do micro, macro and application level experimental studies to get idea on the behavioral performance of full virtualization hypervisors in the hardware experiments. In the first approach, the system was considered as white-box for analyzing its micro-performance by determining bandwidths and latencies of system operations, such as process latencies, IPC latencies, IPC bandwidth and context switching etc.,. In next phase, the system was considered as black box for analyzing its macro-performance based on the memory, processor, disk and network virtualization. Having tested the effectiveness of VM, in order to discover how well the VM performs when serving applications, application benchmarking is carried out as it measures computer system performance as a whole. Our study was motivated by the interests in using virtualization technology in both single multiple virtual machine system. Hence micro benchmark and macro benchmark is carried out in a single virtual machine and application benchmark is carried out in a multiple virtual machine system (server consolidation) with and without workload.

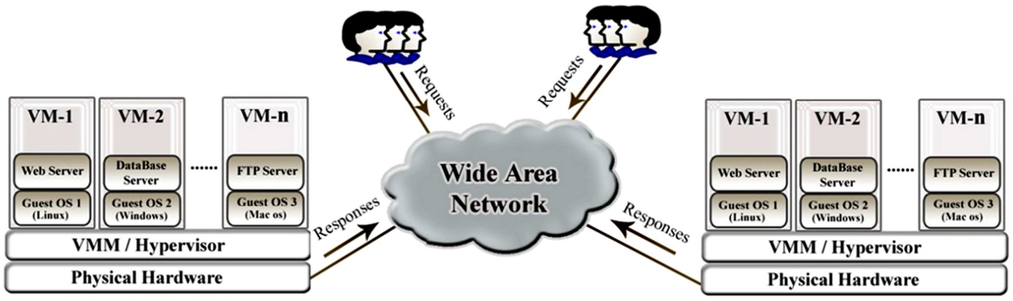

Figure 1: Conceptual Diagram Of Cloud Computing.

4.3 Micro Benchmarks

The benchmarks included in LMbench [22] measure various operating system routines such as context switching, local communications, memory bandwidth, and file operations. Process benchmarks are used to measure the basic process primitives, such as creating a new process, running a different program, and context switching. Process creation benchmarks are of particular interest in distributed systems since many remote operations include the creation of a remote process to shepherd the remote operation to complete. Context switching is important for the same reasons. Interprocess communication latency is important because many operations control messages are to another process (frequently to another system). The time to tell the remote process to do something is pure overhead and is frequently in the critical path of important functions such as distributed applications (e.g., databases, network servers). From the micro-performance data (table 1 and figure 2), we can find that the latencies of process create and context switch in virtualized environment fall behind native environment with huge degree, which implies two main factors that baffle the performance of virtualization systems. Therefore, we may preliminarily determine hardware page table update, interrupt request and I/O are three main performance bottlenecks for common virtualization systems. As most high-cost operations involve them, it's critical for researcher and developer to optimize the handle mechanism of hardware page table update, interrupt request and I/O, etc.

4.3.1 Forkwait

To magnify the differences between the two VMMs, we use the familiar UNIX kernel microbenchmark Forkwait, which stresses process creation and destruction. Forkwait focuses intensely on virtualization-sensitive operations, resulting in low performance relative to native execution. Measuring forkwait, our host required 94 seconds to create and destroy 40000 processes. The KVM on the other hand, took 232 seconds, while the VirtualBox consumed a sobering 624 seconds. Forkwait effectively magnifies the difference between the two VMMs, the VirtualBox inducing approximately 2.69 times greater overhead than the KVM. Still, this program stresses many divergent paths through both VMMs, such as system calls, context switching, creation of address spaces, modification of traced page table entries, and injection of page faults.

Table 1: Kernel Operations Time (Micro Seconds)

	Ubuntu	KVM	VirtualBox
syscall	0.0537	0.0541	0.0596
read	0.1031	0.1056	0.1075
write	0.1251	0.1258	0.1457
stat	0.2484	0.2797	0.2920
fstat	0.0826	0.0929	0.1013
open/close	0.2282	0.2314	0.2532
sigl inst	0.1041	0.1193	0.1247
sigl hndl	0.8588	0.8393	0.8790
pipe	3.4782	3.49	16.4372
fork+exit	287.00	656.875	1785.0564
fork+exec	439.76	690.75	1819.6630
fork+sh	2917.00	4442.72	9057.9412

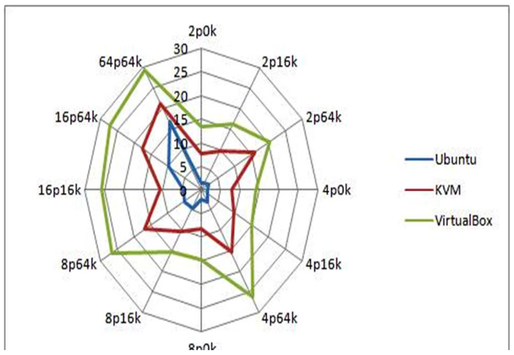

Figure 2: Context Switch Latency (Micro Seconds)

4.4 Macro Benchmarks

The methodology to performance comparison of hypervisors is to drill down each resource component one by one with a specific benchmark workload. The components include CPU, memory, disk I/O, and network I/O. Each component has different virtualization requirements that need to be tested with different workloads. We follow this with a set of more general workloads representative of higher-level applications.

4.4.1 Linux Kernel Compile

The kernel-build benchmark unarchieved the Linux kernel source archieve, and build a particular configuration. It heavily used the disk and CPU. It executed many processes, exercising fork(),exec(),the normal page fault handling code, and thus stressing the memory subsystem; and accessed many files and used pipes, thus stressing the system-call interface. When running on Linux 3.6.5 on x86_64, the benchmark created around 4050 new processes, generated around 24k address space switches (of which 20.4 k were process switches), 4.65M system calls, 3.4M page faults, and between 3.8k and 5.3k device interrupts. We measured and compared benchmark duration and CPU utilization and the result is shown in figure 3 for various numbers of cores. For the single core, KVM completes the job shortly over VirtualBox and for 8 cores there isn't huge difference in completion time.

4.4.2 Bonnie++

One of the major factors in a machine's overall performance is its disk subsystem. By exercising its hard drives, we can get a useful metric to compare VMM instances with, say, virtual Box and KVM guests. Bonnie++ [24] is a disk IO benchmarking tool that can be utilized to simulate a wide variety of different disk access patterns, usually more efficient to define the workload characteristics such as file size, I/O size, and access pattern, simulate a targeted workload profile precisely.

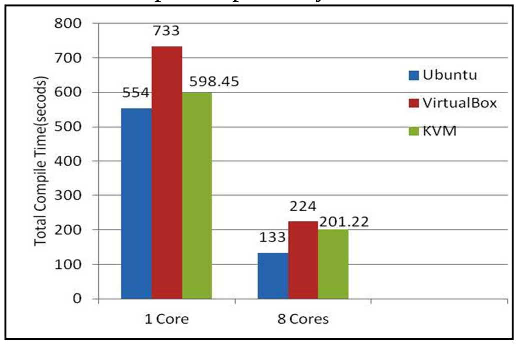

Figure 3: Linux Compile Workloads

Bonnie++ writes one or multiple files of variable size using variable block sizes, attempts to measure both random and sequential disk performance and does a good job simulating real-world loads. Using Bonnie++ to measure the random read, random write and random readwrite performance of a given disk subsystem for a file of 5 GB size at 32 KB I/O size (these characteristics model a simple database) would look as shown in figure 4. In all cases, both VMMs reluctantly accepted in disk usages comparing with native system.

4.4.3 Stream

A simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernel. The stream [23] benchmark has four operating modes: COPY a=b, SCALE a=q*b, SUM a=b+c and TRIAD a=b+q*c. In this test, only the copy mode rely more heavily on the CPU to do some computations on the data being before writing it to memory. This is in contrast to others which measures transfer rates without doing any additional arithmetic; it instead copies a large array from one location to another. The benchmark specifies the array so that it is larger than the cache of the machine and structured so that data reuse is not possible. Table 2 shows the memory performance of two virtual machines for various thread levels. We find the performance of both full virtualizations is very close to the native which means the memory virtualization efficiency is not the bottleneck affecting the performance of cloud applications.

4.4.4 Netperf

Netperf [25] is a network benchmark tool measures the network throughput via TCP and UDP protocols using various packet sizes. The primary foci are bulk (aka unidirectional) data transfer and request/response performance using either TCP or UDP and the Berkeley Sockets interface.

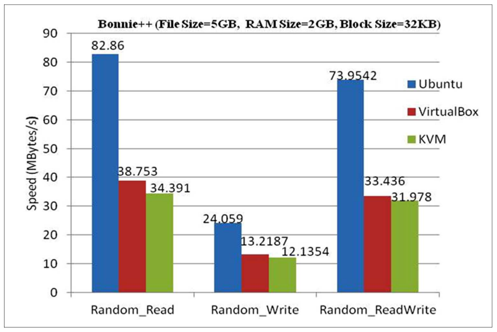

Figure 4: Bandwidth Of Three Disk I/O Operations In Bonnie++

Table 2: The Performance Comparison Of STREAM For Various Numbers Of Cores(Higher Values Are Better)

	2 Threads			4 Threads			8 Threads		
	Ubuntu	**VirtualBox**	**KVM**	**Ubuntu**	**VirtualBox**	**KVM**	**Ubuntu**	**VirtualBox**	**KVM**
Copy	11040.1	10470.3	9152.0	12476.7	11830.1	11429.1	12061.8	11396.2	10724.1
Scale	10581.0	9935.5	8628.4	12435.6	11821.0	11462.3	12046.9	11426.9	10649.8
Add	13919.2	13006.6	11664.0	13911.0	13412.5	12690.3	13478.4	12884.0	12435.1
Traid	13363.9	12432.3	10190.0	13971.8	13331.8	12420.5	13511.6	12804.8	12252.9

A test is based on the netperf TCP_STREAM test that simulates large file transfer such as multimedia streaming and FTP data transfer.Without defining Message Size and SocketSize, the maximum throughput per second is measured from the client using emulated WAN link, where the machines are connected via a 100Mbit connection, and netperf list an actual throughput of 95.13, 91.61, 93.08 Mbits/sec for Ubuntu, VirtualBox and KVM respectively. Whereas, when the experiments are conducted for inter VM communication for VirtualBox and KVM the throughput is 453.29 and 528.64 Mbits/sec respectively. The network throughput of KVM is more than that of VirtualBox in all cases because of QEMU. VirtualBox may have more overhead due to the network transmission using the default bridged network driver located in VMM. As, this requires more levels of indirection compared to KVM hypervisors, which in turn affects overall throughput.

4.5 Application Benchmark Performance Analysis

Having tested the effectiveness of hypervisor, we have to discover how well the VM performs when serving applications. Application benchmarking is a better way of measuring computer system performance as it can present overall system performance by testing the contribution of each component of that system. Hence, we want to be able to benchmark VirtualBox versus KVM virtualization solutions (or bare hardware) for various workloads because each has its strengths and weaknesses compared to another. These application benchmarks will help to determine the best match of the VMM for application. As a single application benchmarks may not be a suitable workload to reveal a VM’s ability in cloud infrastructures, we choose variety of application like httperf[27] for web application, MySQL-SysBench[28] for database workload and POV-RAY[29] for rendering scene workload.

While our previous tests have only considered a single VM running in isolation, it is far more common for each server to run multiple VMs simultaneously. As virtualization platforms attempt to minimize the interference between these VMs, multiplexing inevitably leads to some level of resource contention. That is, if there is more than one virtual machine which tries to use the same hardware resource, the performance of one virtual machine can be affected by other virtual machines. Even though the schedulers in hypervisors mainly isolate each virtual machine within the amount of assigned hardware resources, interference still remains in most of hypervisors. Hence application benchmarks are conducted in single virtual environment and server consolidated environment. In server consolidated environment the experiment is conducted in VM1 with all other virtual machines are running with and without the workload.The workload is generated using stress [26] workload generated tool to determine how the performance degrades as the host's load increases for the various benchmarks. We performed an experiment with single VM as the base case, to check how reactively the algorithms behave towards consolidation with or without workload.

There are three VMs -VM1, VM2 and VM3: VM1 runs as a server (i.e., web server or database server) is being accessed by a client through emulated WAN link, and the other two are used for interference generators using stress tool. The experiment is divided into four phases: first a single VM only runs; In the second phase, all VMs are running with no workload; In the third phase, all VMs are running with VM2 and VM3 ,are the average workload generator, followed by the heavy workload.

4.5.1 Httperf

Httperf [27] is a tool for measuring web server performance that generates HTTP requests and summarizes performance statistics. It supports HTTP and SSL protocols and offers a variety of workload generators. It is designed to run as a single-threaded process using non-blocking I/O to communicate with the server and with one process per client machine, useful to figure out how many users web server can handle before it goes casters-up. It runs on client machines and generates specified number of requests for web-servers in the form of requests per second. The performance characteristics of servers are measured in the form of statistics associated with average response time to a request. By varying the generated workloads, we analyze the server physical resource usage, response time and the comparative result is shown in figure 5. From the figure 5 one can observe, for the various test conditions both VirtualBox and KVM moderately differs and for high workload KVM response time abruptly increasing comparing with VirtualBox.

4.5.2 MySQL-SysBench

SysBench [28] is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system to run a MySQL database under intensive load to evaluate the performance. The idea of this benchmark suite is to quickly get an impression about system performance without setting up complex database benchmarks or even without installing a database at all. SysBench, which was run on a separate client machine, was configured to send multiple simultaneous queries to the MySQL database with zero think time. We used a simple database that fit entirely in memory. As a result, these workloads both saturated the virtual CPU and generated network activity,with little disk I/O.For various number of threads the experiment is conducted and the comparative result of both are given in table 3. KVM works good and equivalent to VirtualBox in many cases, whereas VirtualBox dominates KVM in high load condition. This is because KVM depends on CPU extension whereas VirtualBox doesn't and utilizes the CPU cores well.

4.5.3 POV-RAY

Persistence of Vision Ray-Tracer [29] creates three-dimensional, photorealistic images using a rendering technique called ray-tracing. This renders a standard scene (povray -benchmark) and gives a large number of statistics, ending with an overall summary and rendering time in seconds. The non-virtualized Ubuntu base linux took 830 seconds to render the scence and the comparative result of VirtualBox and KVM is given in table 4. As the load increases both VirtualBox and KVM time taken to render the scene is increasing gradually and KVM works best in this case.

5. RESULTS AND DISCUSSIONS

Our experimental results give a difficult image about the relative performance of these two hypervisors. Clearly, there is no ideal hypervisor that is always the best choice; diverse applications will benefit from different hypervisors depending on their performance needs and the specific features they require. Overall, KVM performs the best in our tests, not surprisingly since KVM is bare-metal architecture and designed based on the hardware virtualization. However, VirtualBox outperforms KVM in certain cases like thread level parallelism and CPU related benchmarks (i.e., using all cores and high load conditions). In general, we find that CPU and memory related tasks experience the lowest levels of overhead, although KVM experiences CPU overheads when all of the system's cores are active. Performance diverges more strongly for disk activities, where both exhibit high overheads when performing all type of disk operations. KVM also suffers in network throughput, but performs much better than VirtualBox. It is worth noting that we test KVM using hardware-assisted full virtualization, whereas the VirtualBox was originally developed for full virtualization.

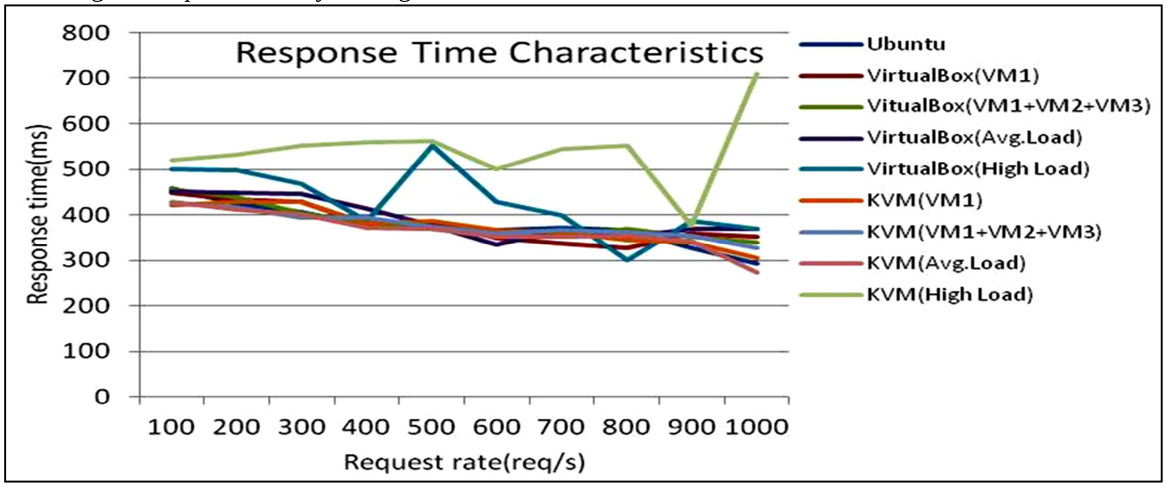

Figure 5: Response time characteristics

Table 3: Mysql-Sysbench Transactions Rate / Second For Server Consolidation With And Without Workload (Higher Values Are Better)

	2 threads		4 Threads		8 Threads	
	VirtualBox	**KVM**	**VirtualBox**	**KVM**	**VirtualBox**	**KVM**
VM1	200.37	215.22	329.06	349.12	137.74	531.56
VM1+VM2+VM3	201.20	217.13	326.57	356.91	137.23	544.71
Avg.Load	164.85	213.55	190.27	277.17	126.23	308.51
High Load	89.05	85.24	105.13	101.60	78.72	82.81

Table 4: POV-RAY Rendering Time (Seconds) For Server Consolidation With And Without Workload

	Virtual Box	**KVM**
VM1	878	817
VM1+VM2+VM3	896	836
Avg. Load	1119	980
High load	1530	1276

Our application level tests match these results, with different hypervisors exhibiting dissimilar overheads depending on the application and the number of cores assigned to them. All of this dissimilarity suggests that properly matching an application to the right hypervisor is complicated, but may well be worth the effort since performance variation is high. We consider that future management systems should be designed to take advantage of this variety. To do so, works needed to overcome the inherent challenges in managing multiple systems with different APIs, and the difficulty in determining what hypervisor best matches an application's needs. Virtual Machine interference also remains a challenge for all of the VMM tested, and is another area where properly designed management systems may be able to help. While we have taken every effort to configure the physical systems and VMs running on them identically, it is true that the performance of each VMM can vary significantly depending on how it is configured. However, this implies that there may be even greater potential for variability between hypervisors if they are configured away from their default settings. Thus the aim of our work is not to definitively show one hypervisor to be better than the others, but to show that each have their own strengths and weaknesses.

6. CONCLUSION AND FUTURE WORK

The primary purpose of this research work was to make the comparison between VirtualBox and KVM. The secondary purpose was to compare the performance of virtualized and non virtualized guests. In the case of a virtualized environment, the abstraction layer between hardware resources and OS is obviously affecting the performance of the virtual guest. VirtualBox and KVM are both different technologies for full virtualization and KVM uses OS layer or paravirtualization approach whereas the VirtualBox uses the hardware layer virtualization. This different approach of virtualization might have created the difference in performance. Experimental results show that: 1) Disk I/O is a performance bottleneck and the latencies of process create and context switch are two main factors that perplex the performance of virtual machine system; 2) The optimized network I/O processing mechanism in KVM can achieve better efficiency compared to VirtualBox since the I/O mechanism of bare metal hardware full virtualization can cause fewer traps than emulated I/O mechanism of VirtualBox ,which performs better performance in inter-domain communication; 3) Different forms of communication overheads (MPI communication, network communication, etc.,) in multiple virtual machine system are the main bottleneck for VirtualBox, which cause huge L2 cache miss rate. Hence one can conclude that different virtualization solution can be implemented within a cloud; The usage of virtualization introduces a degradation of performances because it introduces additional overhead. Virtualization affects CPU usage, network, memory and storage performances as well as applications performances. Within virtualization great performances depend essentially of the tasks scheduling and the workload on the system.

As full virtualizations credit is in running heterogeneous OS environments, we have planned to extend this performance study in various OS. Additionally, the server consolidation experimental works can be carried out with CPU, input/output, Hard Disk Drive and network intensive workload. Further it can be analyzed with various benchmarks and configuration settings, (e.g., KVM only tested using full virtualization I/O mechanism).

REFERENCES:

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica et al., “A view of cloud computing,” Communications of the ACM, Vol. 53, No. 4,2010, pp. 50–58.

[2] L. Wang, G. von Laszewski, A. J. Younge, X. He, M. Kunze, and J. Tao, “Cloud Computing: a Perspective Study,” New Generation Computing, Vol. 28, Mar 2010, pp. 63–69. [Online]. Available: http://cyberaide.googlecode.com/svn/trunk/papers/08-lizhe-ngc/08-ngc.pdf.

[3] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne and J. Matthews, “Xen and the Art of Repeated Research”, *Proceedings of the USENIX 2004 Annual Technical Conference*, FREENIX Track, June 2004,pp. 135-144.

[4] Keith Adams and Ole Agesen,” A comparison of software and hardware techniques for x86 virtualization”, *In ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*,New York, NY, USA, 2006, pp.2–13.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho,R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization”, in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, New York, U. S. A., Oct. 2003, pp. 164–177.

[6] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, “Quantitative comparison of Xen and KVM,” Xen Summit,Boston, MA, USA, 2008, pp. 1–2.

[7] Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W.Zwaenepoel, “Diagnosing Performance Overheads in the Xen Virtual Machine Environment”, *Proceedings of the 1st International Conference on Virtual Execution Environments (VEE 2005)*, June 2005, pp.13-23.

[8] D. S. Ye, Q. M. He, H. Chen, and J. H. Che, “A Framework to Evaluate and Predict Performances in Virtual Machines Environment”, *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008. EUC '08*. Dec. 2008, Vol. 2, pp.375-380.

[9] P.Apparao, R.Iyer,X.Zhang,D.Newell, and Adelmeyer, “Characterization & Analysis of a Server Consolidation Benchmark”, In VEE, 2008.

[10] O.Tickoo., R.Iyer., R.Illikkal., and D. Newell, ”Modeling virtual machine performance: Challenges and approaches”, In HotMetrics, 2009.

[11] Jianhua Che, Congcong Shi, Yong Yu,Weimin Lin, “A Synthetical Performance Evaluation of OpenVZ, Xen and KVM”, *IEEE Asia-Pacific Services Computing Conference*, China, 2010, pp.587-594.

[12] Dawei Huang, Jianhua Che, Qinming He, Qinghua Gao, “Performance Measuring and Comparing of Virtual Machine Monitors”, *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, 2008,pp.381-386.

[13] N. Huber, M. von Quast, M. Hauck, and S. Kounev, ”Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments”, In *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER 2011)*, Noordwijkerhout, The Netherlands, May 7-9, 2011,pp.563 -573..

[14] F. Brosig, F. Gorsler, N. Huber, and S. Kounev, ”Evaluating Approaches for Performance Prediction in Virtualized Environments”, In *Proceedings of the IEEE 21st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2013)*, San Francisco, USA, 2013.

[15] J.Hwang, S.Zeng, F.Wu, and T.Wood, “A component-based performance comparison of four hypervisors”, *Integrated Network Management (IM 2013),2013 IFIP/IEEE International Symposium*,2013, pp.269– 276.

[16] J E. Smith and R. Nair, “The Architecture of Virtual Machines,” *Computer*, Vol. 38, No. 5, 2005, pp. 32–38.

[17] J. Watson, “Virtualbox: bits and bytes masquerading as machines,” *Linux Journal*, Vol. 2008, No. 166, 2008, pp.1.

[18] Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the Linux virtual machine monitor,” in *Proceedings of the Linux Symposium*, Vol.1, July. 2007, pp. 225–230.

[19] G. Neiger, A. Santoni, F. Leung, D. Rodgers,R. Uhlig, “Intel(r) virtualization technology: Hardware support for efficient processor virtualization”, *Intel Technology*

Journal, Vol.10,No.3, August 2006, pp.167 - 178.

[20] AMD: AMD64 Architecture Programmer's Manual Volume 2: System Programming, September 2007.

[21] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols", *ACM SIGCOMM Computer Communication Review*, Vol.27, No.1, Jan. 1997, pp. 31-41.

[22] L. McVoy, C. Staelin, “lmbench: Portable Tools for Perfor-mance Analysis,” *Proceedings of the 1996 USENIX Technical Conference*, San Diego, CA, January 1996, pp.279–295.

[23] McCalpin, D.John., "Memory Bandwidth and Machine Balance in Current High Performance Computers", *IEEE Computer Society Technical Committee on Computer Architecture (TCCA)* Newsletter, December 1995. [Online]. Available: "http://www.cs.virginia.edu/stream/".

[24] Bonnie++, “Disk I/O and file system benchmark”, [Online]. Available: http://www.coker.com.au/bonnie++/

[25] Hewlett-Packard Company, "Netperf: A Network Performance Benchmark," February 1995. [Online]. Available: http://www.netperf.org/netperf/training/Netperf.html.

[26] Stress tool. [Online]. Available: http://weather.ou.edu/_apw/projects/stress/

[27] D. Mosberger and T. Jin, “httperf-a tool for measuring web server performance,” *SIGMETRICS Perform. Eval. Rev.*, Vol. 26, No. 3, Dec. 1998, pp.31–37. [Online]. Available: http://doi.acm.org/10.1145/306225.306235

[28] A.Kopytov,Sysbench, [Online]. Available: http://sysbench.sourceforge.net/.

[29] Andrew S. Glassner, “An Introduction to Ray tracing”, Academic Press 1989, ISBN 0-12-286160-4.