CHAPTER 3

QUALITATIVE AND QUANTITATIVE CASE STUDIES OF RESOURCE INFLUENCE ON SERVER VIRTUALIZATION

3.1 INTRODUCTION

This chapter studies performance behaviours of full virtualization models of different architectures such as hosted (Virtual Box) and bare metal (KVM) virtualization using various benchmarks from micro level, macro level and application level benchmarks with the current hardware advancements and software advancements in the cloud environment. This work compares both virtualization solutions with a base system in terms of application performance, resource consumption, low-level system metrics like context switch, process creation, inter process communication latency and virtualization-specific metrics like virtualization layer consumption. The application benchmarking results reveal that with different hypervisors exhibiting dissimilar overheads depending on the applications and the number of cores assigned to them. In addition, this work provides an analytic modeling that explores the performance implications and limits of server consolidation by the available resources / CPU cores are shared among all VMs. Furthermore, this work studies the limitation of current VMMs and the need of new VMM by briefing the main features of two popular VMM scheduler’s algorithms in specific.
3.2 A PERFORMANCE STUDY OF HARDWARE IMPACT ON FULL VIRTUALIZATION FOR SERVER CONSOLIDATION IN CLOUD ENVIRONMENT

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 para virtualization 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 Virtual Box and KVM. Hence, this sort of performance study often sheds new light on aspects of a work, not fully explored in the previous publications (Clark et al 2004). The objective of this research work is to figure out the following questions: i) The performance degradation of virtual machine against physical machine ii) How much difference between different virtualization technologies? iii) 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.

This chapter work specifically makes the following contributions:
i) Describe tools to measure performance, ranging from the general to the specific, and from the hardware focused to the application oriented
ii) Presents a detailed measurement based performance characterization of the typical server virtualization workloads. iii) Shows VM workload interactions and interference under different degrees of resource sharing. iv) Based on the earlier observations, evaluates two representative full virtualization technologies, Virtual Box and KVM, in various configurations. The work consolidates three more VM systems to drive the system with and without workload. The work 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.

3.2.1 Experimental Methodology

3.2.1.1 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 8 Mbytes ,100 Mbits network card. Both host and virtual machine is configured with 8 VCPUs and 2 GB RAM, 60 GB HDD with Ubuntu 13.10 (Saucy Salamander). The virtualization solutions considered are KVM 76, Virtual Box 4.2. In all solutions we use hardware virtualization support to virtualize 64-bit guests over a 64-bit host. For the KVM and Virtual Box machines, virtual NICs are using the default bridged network driver. The cloud environment can be emulated using
Dummynet (Huang et al 2013; Rizzo 1997) and the experimental setup mimic as shown in the Figure 3.1. 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.

![Figure 3.1 Conceptual Diagram of Cloud Computing (Jamal et al 2009)](image)

### 3.2.2 Benchmarks

We do micro level, macro level and application level experimental studies to get the through 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, Inter Process Communication (IPC) latencies, IPC bandwidth, context switching etc. In the next phase, the system was considered as black box for analyzing its macro-performance on the memory virtualization, processor virtualization, disk virtualization 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.

The work was motivated by the interests in using virtualization technology in both single virtual machine system and multiple virtual machines system. Hence micro benchmark and macro benchmark is carried out in single virtual machine and application benchmark is carried out in multiple virtual machine system (server consolidation) with or without workload.

3.2.2.1 Micro benchmarks

The benchmarks included in LMbench 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 (McVoy & Staelin 1996). 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 completion. Context switching is important for the same reasons. Interprocess communication latency is important because many operations are control messages to another process (frequently on 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 as shown in Table 3.1, it is observed that the latencies of process create and context switch in virtualized environment fall behind native environment with a huge degree, which implies two main factors that baffle the performance of virtualization systems. Therefore, the work 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.

Table 3.1 Kernel Operations Time and Context Switch Latency in μs

<table>
<thead>
<tr>
<th></th>
<th>Ubuntu</th>
<th>KVM</th>
<th>VirtualBox</th>
<th>Context Switch</th>
<th>Ubuntu</th>
<th>KVM</th>
<th>VirtualBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscall</td>
<td>0.0537</td>
<td>0.0541</td>
<td>0.0596</td>
<td>2p0K</td>
<td>1.40</td>
<td>7.66</td>
<td>13.275</td>
</tr>
<tr>
<td>read</td>
<td>0.1031</td>
<td>0.1056</td>
<td>0.1075</td>
<td>2p16K</td>
<td>1.59</td>
<td>9.61</td>
<td>15.975</td>
</tr>
<tr>
<td>write</td>
<td>0.1251</td>
<td>0.1258</td>
<td>0.1457</td>
<td>2p64K</td>
<td>2.09</td>
<td>15.75</td>
<td>20.268</td>
</tr>
<tr>
<td>stat</td>
<td>0.2484</td>
<td>0.2797</td>
<td>0.2920</td>
<td>4p0K</td>
<td>1.62</td>
<td>7.84</td>
<td>14.118</td>
</tr>
<tr>
<td>fstat</td>
<td>0.0826</td>
<td>0.0929</td>
<td>0.1013</td>
<td>4p16K</td>
<td>1.76</td>
<td>9.56</td>
<td>14.855</td>
</tr>
<tr>
<td>open/close</td>
<td>0.2282</td>
<td>0.2314</td>
<td>0.2532</td>
<td>4p64K</td>
<td>3.00</td>
<td>15.25</td>
<td>26.190</td>
</tr>
<tr>
<td>sigl inst</td>
<td>0.1041</td>
<td>0.1193</td>
<td>0.1247</td>
<td>8p0K</td>
<td>2.10</td>
<td>8.39</td>
<td>14.872</td>
</tr>
<tr>
<td>sigl hndl</td>
<td>0.8588</td>
<td>0.8393</td>
<td>0.8790</td>
<td>8p16K</td>
<td>4.57</td>
<td>10.36</td>
<td>15.160</td>
</tr>
<tr>
<td>pipe</td>
<td>3.4782</td>
<td>3.49</td>
<td>16.44</td>
<td>8p64K</td>
<td>5.20</td>
<td>16.85</td>
<td>26.736</td>
</tr>
<tr>
<td>fork+exit</td>
<td>287.00</td>
<td>656.85</td>
<td>1785.06</td>
<td>16p16K</td>
<td>4.53</td>
<td>10.65</td>
<td>25.830</td>
</tr>
<tr>
<td>fork+exec</td>
<td>439.76</td>
<td>690.75</td>
<td>1819.66</td>
<td>16p64K</td>
<td>9.74</td>
<td>17.50</td>
<td>27.28</td>
</tr>
<tr>
<td>fork+sh</td>
<td>2917.00</td>
<td>4442.72</td>
<td>9057.94</td>
<td>64p64K</td>
<td>16.56</td>
<td>21.06</td>
<td>29.18</td>
</tr>
</tbody>
</table>

3.2.2.1 Forkwait

To magnify the differences between the two VMMs, we use the familiar UNIX kernel micro benchmark 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 Virtual
Box consumed a sobering 624 seconds. ForKwait effectively magnifies the difference between the two VMMs, the Virtual Box 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.

3.2.2.2 Macro benchmarks

The methodology for our 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. The work follows this with a set of more general workloads representative of higher-level applications.

3.2.2.2.1 Linux kernel compile

The kernel-build benchmark unarchieved the Linux kernel source archive, 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 24 Kilo (K) address space switches (of which 20.4K were process switches), 4.65 Mega (M) 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 3.2 for various numbers of cores. For the single core, KVM completes the job shortly over Virtual Box and for 8 cores there isn’t huge difference in their completion time.
3.2.2.2 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 benchmark has four operating modes: COPY a=b, SCALE a=q*b, SUM a=b+c and TRIAD a=b+q*c (McVoy & Staelin 1996). 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 3.2 shows the memory performance of two virtual machines for various thread levels. The work finds 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.
Table 3.2  The Performance Comparison of STREAM for Various Numbers of Cores (Higher Values are Better)

<table>
<thead>
<tr>
<th></th>
<th>2 Threads</th>
<th>4 Threads</th>
<th>8 Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ubuntu</td>
<td>VirtualBox</td>
<td>KVM</td>
</tr>
<tr>
<td>Copy</td>
<td>11040.1</td>
<td>10470.3</td>
<td>9152</td>
</tr>
<tr>
<td>Scale</td>
<td>10581.0</td>
<td>9935.5</td>
<td>8628.4</td>
</tr>
<tr>
<td>Add</td>
<td>13919.2</td>
<td>13006.6</td>
<td>11664</td>
</tr>
<tr>
<td>Traid</td>
<td>13363.9</td>
<td>12432.3</td>
<td>10190</td>
</tr>
</tbody>
</table>

3.2.2.2.3 Bonnie++

One of the major factors in a machine’s overall performance is its disk subsystem. By exercising its hard drives, the work get a useful metric to compare VMM instances with, say, virtual Box and KVM guests. Bonnie++ 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 (Bonnie++ 2012). 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 5GB size at 32 KB I/O size (these characteristics model a simple database) would look as shown in Figure 3.3. In all cases, both VMMs reluctantly accepted in disk usages comparing with native system.
3.2.2.4 Netperf

Netperf is a network benchmark tool that 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. A test is based on the netperf TCP_STREAM test that simulates large file transfer such as multimedia streaming and FTP data transfer (Netperf 1995). 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 lists an actual throughput of 95.13, 91.61, 93.08 Mbits/sec for Ubuntu, Virtual Box and KVM respectively. Whereas, when the experiments are conducted for inter VM communication for Virtual Box and KVM the throughput is 453.29 and 528.64 Mbits/sec respectively. The network throughput of KVM is more than that of Virtual Box in all cases because of QEMU. Virtual Box 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.

3.2.2.3 Application benchmark performance analysis

Having tested the effectiveness of hypervisor, the work has 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, the work want to be able to benchmark Virtual Box Vs. KVM virtualization solutions (or bare hardware) for various workloads because each has both strengths and weaknesses compared to one 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 httpperf (Mosberger & Jin 1998) for web application, MySQL-SysBench (Kopytov 2004) for database workload and POV-RAY (Andrew & Glassner 1989) for rendering scene workload.

While the 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 tool (2012) workload generated tool to determine how the performance degrades as the host’s load increases for the various benchmarks. The work 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: one VM 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.

3.2.2.3.1 Httperf

Httperf (Mosberger & Jin 1998) 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, this work analyze the server physical resource usage, response time and the comparative result is shown in Figure 3.4. From the Figure 3.4
one can observe, for the various test conditions both Virtual Box and KVM moderately differs and for high workload KVM response time abruptly increasing comparing with Virtual Box.

![Response Time Characteristics](image)

**Figure 3.4 Response Time Characteristics of Virtual Box and KVM**

### 3.2.2.3.2 MySQL-SysBench

SysBench 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 its 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 (Kopytov 2004). SysBench, which was run on a separate client machine, was configured to send multiple simultaneous queries to the MySQL database with zero think time. The work used a simple database that fit entirely in memory. As a result, these workloads both saturated the virtual CPU and generated significant network activity, with very little disk I/O. For various numbers of threads the experiment is conducted and the comparative results of both are given in Table 3.3. KVM works good and equivalent to Virtual Box in many cases, whereas Virtual
Box dominates KVM in high load condition. This is because KVM depends on CPU extension whereas Virtual Box doesn’t and utilizes the CPU cores well.

**Table 3.3 MySQL-SysBench Transactions Rate / Second for Server Consolidation with and without Workload (Higher Values are Better)**

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>2 threads</th>
<th>4 threads</th>
<th>8 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VirtualBox</td>
<td>KVM</td>
<td>VirtualBox</td>
</tr>
<tr>
<td>VM1</td>
<td>200.37</td>
<td>215.22</td>
<td>329.06</td>
</tr>
<tr>
<td>VM1+ VM2+ VM3</td>
<td>201.20</td>
<td>217.13</td>
<td>326.57</td>
</tr>
<tr>
<td>Avg. Load</td>
<td>164.85</td>
<td>213.55</td>
<td>190.27</td>
</tr>
<tr>
<td>High Load</td>
<td>89.05</td>
<td>85.24</td>
<td>105.13</td>
</tr>
</tbody>
</table>

**3.2.2.3.3 POV-RAY**

Persistence of Vision Ray-Tracer 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 (Andrew & Glassner 1989). The non-virtualized Ubuntu base Linux took 830 seconds to render the scene and the comparative result of Virtual Box and KVM is given in Table 3.4. As the load increases both Virtual Box and KVM time taken to render the scene is increasing gradually and KVM works best in this case.
Table 3.4  **POV-RAY Rendering Time (Seconds) for Server Consolidation with and without Workload**

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Virtual Box</th>
<th>KVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM1</td>
<td>878</td>
<td>817</td>
</tr>
<tr>
<td>VM1+VM2+VM3</td>
<td>896</td>
<td>836</td>
</tr>
<tr>
<td>Avg. Load</td>
<td>1119</td>
<td>980</td>
</tr>
<tr>
<td>High load</td>
<td>1530</td>
<td>1276</td>
</tr>
</tbody>
</table>

### 3.2.3  **Results and Discussion**

The 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, Virtual Box outperforms KVM in certain cases like thread level parallelism and CPU related benchmarks (i.e., using all cores and high load conditions). In general, this work finds 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 Virtual Box. It is worth noting that the work test KVM using hardware-assisted full virtualization, whereas the Virtual Box was originally developed for full virtualization.
The 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. This work considers 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 this work 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 this 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.

This different approach of virtualization might have created the difference in performance. Experimental results show that: i) 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; ii) The optimized network I/O processing mechanism in KVM can achieve better efficiency compared to Virtual Box since the I/O mechanism of bare metal hardware full virtualization can cause fewer traps than emulated I/O mechanism of Virtual Box, which performs better performance in
inter-domain communication; iii) Different forms of communication overheads (MPI communication, network communication, etc.,) in multiple virtual machine system are the main bottleneck for Virtual Box, 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.

3.3 RESOURCE INFLUENCE ON APPLICATION PERFORMANCE IN SERVER VIRTUALIZATION

As virtualization introduces dynamics and increases flexibility, a variety of additional factors can influence the performance of virtualized systems. Huber et al (2011b) and Armbrust et al (2010) having analyzed major representative virtualization platforms, abstracted a generic performance model of VM performance influencing factors as shown in Figure 3.5. Those are virtualization type, hypervisor’s architecture, resource management configuration and workload profile. Though, several influencing factors are grouped under the resource management configuration, the CPU scheduling configuration has a significant influence on the virtualization platform’s performance and chief among them are virtual CPUs allocated to a VM, the number of VMs and resource over commitment. Managing virtual CPU requires an additional management layer in the hypervisor and the number of VMs has a direct effect on how the available resources are shared among all VMs.
As performance provision is the major concern of VMM scheduler in cloud for the user acceptance, this section provides i) Two quantitative case studies that focus on how the VM performance is affected by adapting a VM’s CPU capacity and ii) Two qualitative case studies of widely popular virtualization VMM schedulers.

### 3.3.1 Analytical Model

As improved performance is pushing the migration to multi-core processors, this section explores power and performance implications and limits of server consolidation through a simplified model. It provides an analytic modeling to evaluate the number of VMs that has a direct effect on how the available resources (CPU cores) are shared among all VMs. Assume, as if consolidating applications running on $m$ Physical Machine (PM) of capacity $C$ onto one PM is given as $nC$. The paper utilizes an M/G/1 queuing model (Kleinrock 1975) to calculate the response time (RT) exhibited when processing requests a function of computational capacity and request arrival rate (Almeida & Menasce 2002). The model assumes an exponentially
distributed request inter arrival time with mean $1/\lambda$, and a server which process requests with a constant service time with mean $1/\mu$. Based upon these assumptions, the model defines the average normalized response time = (average response time of an algorithm / average response time of the baseline algorithm), as $RT = \frac{T}{\lambda} (T$ - the average time spent in the system). The queuing theory yields the normalized response time ($RT_o$) using $m$ PM as,

$$RT_o = \frac{\rho}{1 - \rho} \quad (3.1)$$

for each of the PM (where $\rho$(server workload) = $\lambda/\mu$). When consolidating these VMs into one PM ($m$ processors), the queue becomes one with $m$ machines working at the same rate $\mu$, and servicing an arrival rate of $m*\lambda$. Subsequently, the normalized response time ($RT_p$) for one server with $p$ processors is

$$RT_p = \frac{m*\rho + Q_P}{(1 - \rho)} \quad (3.2)$$

(Here $Q_p$ is the ‘queuing probability’, $Q_p <<$ for light loads). Suppose, a single PM that is $m$ times faster, once again it is modeled as a single server queue but with service rate $m* \mu$. Since $\rho$ remains intact, the normalized RT ($RT_c$) remains the same as $RT_o$ in (3.1). Thus, it is apparent that for $Q_p <<$ case, the consolidation onto a multi-core machine versus fat CPU can result significant degradation in performance, at least equal to average normalized RT and, the second term is poor in both cases for heavy loads (Calheiros et al 2011).

Now, consider the case where the single PM onto which the workload was consolidated is only $n$ times faster than the original servers. In this case, it is obvious that the normalized response time $RT_n$ is,

$$RT_n = \frac{m*\rho}{(n-m*\rho)} \quad (3.3)$$
From the Equation (3.3), it is apparent that it is possible to use a PM far less powerful than the consolidation of the \( m \) original PM, as long as \( n/m \) remains reasonably large as compared to \( \rho \); and if indeed \( n \gg m \cdot \rho \) then the average normalized RT degrades only linearly by the factor of \( n/m \).

Thus, it is obvious that the analysis yields a few limits to the server aggregation using virtualization. The theoretical maximum benefit, in terms of a reduction in number of servers, is \( n/m = \rho \), where the system becomes unresponsive. In practice, it is possible to get results close to this, i.e., if \( n/m = \rho \ (1+ \epsilon) \), then the average normalized RT becomes \( 1/\epsilon \). In consequence, the initial inefficiency can be decided on an acceptable average normalized RT. It paves space to plan the consolidation strategy accordingly.

Subsequent to the aforementioned case, consolidation onto an \( n \)-core server, and average normalized RT \( \text{RT}_p \), can be given as:

\[
\text{RT}_p = m\rho + Q_p \left( 1 - \frac{(m\rho)}{n} \right)
\]  

(3.4)

It is apparent that the RT remains the same as \( \text{RT}_p \) in Equation (3.2) for \( Q_p \ll \). Thus, the RT is decreased by a factor of \( m \), and becomes independent of \( n \), as compared to fat CPU case (1). However, in the case of \( Q_p \gg \) heavy load, the second term dominates, as there is a degradation in the performance in the multi-core case \( n \ll m \) over \( m = n \), say Equation (3.2).

When \( Q_p \ll \), the response time is same as Equation (3.2) shows that RT is decreased by a factor of \( m \), independent of \( n \) over Equation (3.1). In addition, when \( Q_p \gg \), is a marked degradation in performance in the multi core processor case if \( n \ll m \), as compared to Equation (3.2). It directs the end that the cores allocated to a VM directly impact the hosted application’s performance. Further, analytical model proves that multithreaded application
servers can exploit multi-core architectures efficiently. Thus clouds rely almost exclusively on multi-core systems.

As power consumption of chips is given as \( P \propto V^2 \) (\( P \)-power, \( V \)-voltage), a system that runs at a clock speed \( n \) times faster than a ‘base’ system, will consume \( n^2 \) the power of the base system. Thus, it is apparent that the multi core (\( n \) core) system consumes only \( n \) times the power. Thus, there is a trade off between reducing the power usage by VMs aggregation onto the multi core CPU systems, versus improved performance on systems with fat CPU but at the cost of nonlinear growth in power consumption per PM.

3.3.2 Performance Study

3.3.2.1 Impact of CPU allocation

As it was discussed earlier, CPU cores are one of the main sources of performance interference as shown in Figure 3.6. Even with such physical isolation for the CPU, the typical relationship between application performance and the CPU allocation is difficult. Thus, this complexity is demonstrated by executing two types of experiments, targeted at the component and application level in virtual machine environment by setting CPU limit at different levels.

The performance of the CPU intensive applications (kernel compilation) and virtualized applications (OLTP) are measured while varying the VM’s CPU limit from 1 core to 8 cores. All resource allocations were kept high enough to ensure that those are all not the bottleneck. All the experiments were conducted on physical hardware configured with AMD FX 8-Core Black Edition FX-9590. It has 8 *4.7 GHz AMD Opteron 8 core processors with 3 MB L2, 6 MB L3 cache each, 8 GB DDR2-667 main
memory, 100 GB of storage and 10/100/1000-BASE-T Ethernet connections. Both host and virtual machine are configured with 8 VCPUs and 4 GB RAM, 50GB HDD with Ubuntu 14.04 LTS (Trusty Tahr).

The virtualization solutions considered for the experiment is Xen 5.0. In all solutions, hardware virtualization support is used to virtualize 64-bit guests over a 64-bit host. For the Xen machines, virtual NICs use the default bridged network driver. Two types of benchmarks (Ciliendo & Kunimasa 2007) namely Linux kernel compile, MySQL-SysBench are used and are targeted at the component and application level of influencing factors.

a) Linux kernel compile: The kernel build benchmark unarchived the Linux kernel source archive, 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. It accessed many files and used pipes, thus stressing the system call interface.

b) MySQL SysBench: It 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 its performance. SysBench, which was run on a separate client machine, was configured to send multiple simultaneous queries to the MySQL database with zero think time. A simple database that fit entirely in memory is used. As a result, these workloads both saturated the virtual CPU and generated network activity, with very little disk I/O.

For various numbers of threads the experiment is conducted and the results of both are given in the Figure 3.6 and Figure 3.7. It shows the normalized performance of these examinations. As seen from the graph, both the benchmarks workloads behave linearly, and the performance slope is different at various CPU allocation ranges. While the kernel compilation saturate quickly at 3 VCPUs, SysBench performance, on the other hand,
varies almost linearly with CPU allocation. But at some time the saturated point is visible because of resource over provisioning. Thus, this data reveals the fact that virtualized workloads can have quite different performance curves with respect to number of CPU allocation.

![Figure 3.6 The Performance Impact of Core Mappings](image1)

![Figure 3.7 SysBench Database Transactions](image2)

The above analysis of hypervisor’s behaviors demonstrates that resource pools are one of the vital factors in the constitution of virtualization overhead and current scheduling scheme in conventional VMM the shows
bottlenecks on the massive advanced system with heavier load i.e., more VMs and heavier stress as shown in the experiment. Hence, in order to maximum the hardware resource utilization, VM management has become an important research field of virtualization technology. Thus, VM scheduling is crucial for the throughput of a system and affects the overall system performance.

This leads to the conclusions that resources allocated to a virtual machine directly have an impact on the hosted application’s performance and choosing appropriate control knobs to handle resource allocation for a VM is critical to ensure desirable performance and create a robust model.

3.4 STATE OF THE ART VMM SCHEDULERS

VCPU scheduling remains as a challenge for Virtualization technologies, especially with hypervisors starting to host Chip Multithreading VMs. A naive, yet popular, implementation is to use a simple Round-Robin algorithm when assigning processor resources to each VCPU. This option is available in most hypervisors. e.g. in KVM or Virtual Box hypervisors. However, this approach can cause additional synchronization latency for guest VMs due to VCPU preemption. Whereas, VMware ESX and Xen are two of the leading virtualization systems for the x86 architecture, and they both allow for CMT virtualization. However, implementing CMT virtualization is difficult because the two technologies have different goals, and virtualization in particular can conflict with the expected behaviour of a CMT system. As the implemented prototype in this work is a generic, it discusses briefing the main features of these two VMM scheduler’s algorithms in specific.
3.4.1 CPU Scheduling Algorithms in Xen

Xen is quite unique among VM platforms because it allows user to choose among different CPU schedulers. It implements a higher level abstraction scheduling operations, where each scheduler needs to implement its own scheduling policy and registers itself to this interface. Xen supports three different types of schedulers namely Borrowed Virtual Time (BVT), Simple Earliest Deadline First (SEDF) and Credit Scheduler. The users can set the scheduler option during Xen’s boot time by passing the parameter value of scheduler.

(i) Borrowed Virtual Time (BVT) - It is a proportional share scheduler that is suited for I/O intensive domains. The scheduler adjusts itself dynamically with the varying I/O intensities when specified with the correct parameters. It is based on the concept of virtual time, dispatching the runnable VM with the smallest virtual time the low latency support is provided in BVT for real time and interactive applications by allowing latency sensitive client to warp back in virtual time to gain scheduling priority. And the client can effectively borrow virtual time from its future CPU allocation. Each runnable domain Dom\textsubscript{i} will receive CPU proportion according to its weight w\textsubscript{i}, and the virtual time vt\textsubscript{i} of Dom\textsubscript{i} is incremented by its running time rt\textsubscript{ij} in the j\textsuperscript{th} scheduling around, divided by w\textsubscript{i}: vt\textsubscript{i} ← vt\textsubscript{i} + rt\textsubscript{ij} / w\textsubscript{i}. However, due to the lack of Non Work Conserving (NWC) mode (unused CPU cycles of one domain can’t be used by the other domain), its usage is severely limited in many application environments.

(ii) Simple Earliest Deadline First (SEDF) - In this algorithm, the domains request a minimum time slice that requires for
communication. The request is a tuple of \((s_i, p_i, x_i)\), which means \(Dom_i\) will receive \(s_i\) units of time in each period of length \(p_i\). The \(x_i\) is a boolean flag indicating whether \(Dom_i\) is scheduled in WC-mode or NWC-mode. SEDF performs well when the workload is low, but when running in heavy workload, many clients are observed to miss their deadlines and the scheduling overhead significantly increases, where the domain requests for ‘t’ slices every ‘p’ periods of CPU time. One main shortage is the lack of global workload balancing on multiprocessors, and the CPU fairness depends on the value of the period. Besides, the lack of global load balancing on multiprocessors, implementation also limits its usage.

(iii) Credit Scheduling - BVT lacks NWC-mode while SEDF is found to be unstable under heavy workload and does not support CMT well, so both of them were replaced by Credit scheduler in Xen. The credit based scheduler is recently incorporated into Xen and it provides better load balancing and low latency mechanisms. This algorithm is a kind of Proportional Share (PS) strategy, featuring automatic workload balancing of virtual CPUs across physical CPUs on a CMT host. According to the scheduling algorithm of Credit Scheduler using in Xen hypervisor, each virtual CPU is asynchronously assigned to a physical CPU by CPU scheduler in order to maximize the throughput. Specifically, when there is no runnable VCPU on the current physical CPU, the scheduler will try to migrate one runnable VCPU from the other physical CPUs. Each domain is assigned with a (weight, cap) pair.
Similarly, the scheduler allocates CPU time proportion (in credit) to each domain according to its weight. All queued VCPUs are sorted by their remaining credit, and the scheduler will select the VCPU that has most credit to run. When the cap is 0, VM receives extra physical CPU (WC-mode), while a nonzero cap (expressed as a percentage) limits the amount of physical CPU time obtained by a VM (NWC-mode). The algorithm uses followers interval for the physical CPU allocation. The priorities (credits) all runnable VMs which are recalculated in the interval, which is mainly in proportion to weight those VMs, are assigned by the user. This algorithm can efficiently achieve a global workload balancing on a CMT system when the majority of the workload is not the high concurrent application. However, all these choice come with the burden of choosing the right scheduler and configuring it (Cherkasova et al 2007; Chisnall et al 2007).

3.4.2 VMware ESX Server VCPU Scheduling Algorithms

The default approach by KVM or Virtual Box hypervisors (Round-Robin algorithm) cause additional synchronization latency for guest virtual machines due to VCPU preemption. In order to eliminate this synchronization latency, VMware applies a co-scheduling algorithm (Sukwong & Kim 2011), which uses a concept similar to gang scheduling (Schwiegeishohn & Yahyapour 1998). Co-scheduling requires that all VCPUs are associated with a VM to be scheduled simultaneously in order for the VM to run. Such an algorithm helps to avoid the synchronization latency, as both the waiting VCPUs and the lock holding VCPU are preempted and resumed at the same time.

This “strict” co-scheduling approach, however, introduces a fragmentation problem. A VCPU can only be scheduled after the hypervisor gathers enough resources to execute all other VCPUs in the same VM.
However, ESX has several optimizations to improve performance over a naive implementation of co-scheduling, which would require even idle VCPUs in a VM to execute. First, ESX is able to detect if a VCPU is executing an idle loop, and in this case ESX does not schedule an idle VCPU to run nor require it to be co-scheduled for active VCPUs to run. Second, ESX uses a technique called relaxed co-scheduling that helps prevent requiring physical CPUs from being idle in order to start running VCPUs in an CMT system. ESX provides three control knobs for CPU allocation to individual VMs: reservation, limit, and shares. Reservation guarantees a certain minimum CPU allocation expressed in MHz Limit (in MHz) provides an upper bound on the CPU allocation. Share provides a mechanism for proportional allocation during time periods when the sum of the CPU demands of the currently running VMs exceeds the capacity of the physical host.

3.5 CONCLUSION

As performance study is an ongoing pursuit and 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 work studied performance behaviours of full virtualization models of different architectures such as hosted (Virtual Box) and bare metal (KVM) virtualization using various benchmarks from micro level, macro level and application level benchmarks with the current hardware advancements and software advancements in the cloud environment. This work compared 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. Micro level, macro level and application level used to identify the effectiveness of
the experiments, yields that Virtual Box outperforms KVM in CPU and thread level parallelism and KVM outperforms in all other cases. Both are very reluctantly accepted in disk usages comparing with native system.

Having analyzed two major representative virtualization platforms, one can infer that current commercial resource management tools provide only partial solutions to VCPU scheduling problem i.e., it provides resource management capabilities by forcing virtual machines allocation to be within certain limits. In addition, these tools do not address setting these limits with appropriate values for each application, or how they should be changed in case. Thus, a resourceful VMM scheduler is important for increased throughput and decreased response time. Given varying workloads, there is a particular scheduling algorithm that is more efficient at scheduling VM for particular types of workloads. Thus, it is possible to fine tune the VMM scheduler to maximize throughput and minimize response time with specific type of workloads subject to SLA.

The inference of the chapter is multi core machine CPU is currently the most deployed type of hardware architecture for high performance computing. System virtualization on the other hand is increasingly adopted for various reasons. In a virtualized multi core machine system, the multi core machine attributes are transparent to guest OS’s. Thus, a Virtual Machine Monitor (VMM) is required to have multi core machine aware resource management. Tradeoffs between performance, power, and energy are observable as virtual cores (vcores) and / or virtual addresses are mapped in different ways.