Vol. 4, No. 12, December 2013 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2009-2013 CIS Journal. All rights reserved.
http://www.cisjournal.org

Benchmarking the Performance of Microsoft Hyper-V server, VMware ESXi and Xen Hypervisors

[1] Hasan Fayyad-Kazan, [2] Luc Perneel, [3] Martin Timmerman
[1] PhD Candidate, Department of Electronics and Informatics, Vrije Universiteit Brussel, Pleinlaan 2- 1050 Brussels, Belgium
[2] PhD Candidate, Department of Electronics and Informatics, Vrije Universiteit Brussel, Pleinlaan 2- 1050 Brussels, Belgium
[3]Professor, Department of Electronics and Informatics, Vrije Universiteit Brussel, Pleinlaan 2- 1050 Brussels, Belgium
[3]Dedicated-Systems Experts, Diepenbeemd 5, 1650 Beersel, Belgium

[1] hafayyad@vub.ac.be, [2] luc.perneel@vub.ac.be, [3] martin.timmerman@vub.ac.be , [3] martin.timmerman@dedicated-systems.com

ABSTRACT

Virtualization is a technology that allows the physical machine resources to be shared among different virtual machines (VMs). It is provided by a software layer called hypervisor or Virtual Machine Monitor (VMM). The hypervisor abstracts the hardware from the operating system permitting multiple operating systems to run simultaneously on the same hardware. Many different hypervisors--both open source and commercial-- exist today and are widely used in parallel and distributed computing. While the goals of these hypervisors are the same, the underlying technologies vary. Our target hypervisors in this paper are: Microsoft (MS) Hyper-V Server, VMware vSphere ESXi and Xen. MS Hyper-V Server and Xen are micro-kernelized hypervisors which leverage para-virtualization together with full-virtualization/hardware-assisted virtualization approaches, while ESXi is a monolithic hypervisor supporting only full-virtualization/hardware-assisted approach.
A series of performance experiments are conducted on each virtualization approach of the latest versions (at the time of this study which started in May 2013) for the mentioned hypervisors using Linux PREEMPT-RT as the guest operating system. This paper discusses and compares the results of these experiments.

Keywords: *ESXi, Full-virtualization, Hyper-V, Para-virtualization, Virtualization, Xen*

1. INTRODUCTION

Virtualization is an emerging technology which provides organizations with a wide range of benefits [1]. It is seen as an efficient solution for optimum use of hardware, improved reliability and security [2]. It has transformed the thinking from physical to logical, treating IT resources as logical resources rather than separate physical resources [3]. In simple words, it is a technology that introduces a software abstraction layer between the underlying hardware i.e. the physical platform/host and the operating system(s) (OS) i.e. the guest virtual machine(s) (VM) including the applications running on top of it. This software abstraction layer is known as the virtual machine monitor (VMM) or a hypervisor [1].

Hypervisors are generally grouped into two classes: Type 1 hypervisors run directly on the system hardware and thus are often referred to as bare-metal hypervisors; Type 2 hypervisors run within a conventional operating-system environment and are referred to as hosted hypervisors.

Since it has direct access to the hardware resources rather than going through an operating system, a bare-metal hypervisor is more efficient than a hosted architecture and delivers greater scalability, robustness and performance [4]. In this work, three bare-metal hypervisors' families are used: MS Hyper-V Server, VMware vSphere ESXi and Xen.

Moreover, (Type 1) bare-metal hypervisors are divided into two subcategories: Monolithic and Micro-kernelized designs. The difference between them is the way of dealing with the device drivers.

MS Hyper-V Server and Xen are micro-kernelized hypervisors which leverages para-virtualization together with full-virtualization, while VMware ESXi is a monolithic hypervisor which leverages hardware emulation (full-virtualization) [5].

This paper provides a quantitative performance comparison between the virtualization approaches supported by each hypervisor. It is organized as follows: Section 2 is a theoretical explanation about the difference between Monolithic and Micro-kernelized hypervisors; Section 3 explains briefly the virtualization approaches; Section 4 describes the architectures of the three used hypervisors; section 5 is a detailed explanation about the test methodology, scenarios and results obtained; and section 6 gives a final conclusion.

2. MONOLITHIC VS. MICRO-KERNALIZED

The methodology for how a bare-metal hypervisor allocates available resources, and how it handles driver usage, depends on whether the hypervisor is a Micro-kernalized or Monolithic Hypervisor [6].

Monolithic hypervisor (figure 1), such as VMware ESXi, handles all the hardware access for each VM. It hosts drivers for all the hardware (storage, network, and input devices) that the VMs need to access [7]. Having all device drivers reside within the hypervisor, VMware has to be very picky about what systems will support their hypervisor and which ones will not. Therefore, VMware ESXi will only run on a selective number of systems [6].

The biggest advantage of this design is that it does not require a host operating system. The hypervisor acts as the operating platform that supports all the virtual

©2009-2013 CIS Journal. All rights reserved.

operating systems running on the hardware [8]. This design's drawbacks are: limited hardware support and instability--the device drivers are directly incorporated into the layers of functionality which means that if one driver is hit by an update, bug, or security vulnerability, the entire virtual architecture within that physical machine will be compromised [8].

In contrast, Micro-kernelized design (figure 1) – Hyper-V Server and Xen– does not require the device drivers to be part of the hypervisor layer. Instead, drivers for the physical hardware are installed in the operating system, which is running in the parent partition (VM) of the hypervisor [9]. This means that there is no need to install the physical hardware supporting device drivers for each guest operating system running as a child partition, because when these guest operating systems need to access physical hardware resources on the host computer, they simply do this by communicating through the parent partition [10]. This communication can be via a very fast memory-based bus in case the child partition is para-virtualized, or using the emulated devices provided by the parent partition in case of full-virtualization.

This design's advantages are: No need to incorporate the device drivers in the hypervisor kernel; device drivers do not need to be hypervisor-aware which results in a wide range of devices used with the Micro-kernelized hypervisor [9].

Its drawback is the requirement to install an operating system in the parent partition before the hypervisor can operate. Moreover, if this operating system crashes for some reason (update, bug, etc.) then all other VMs will crash [9].

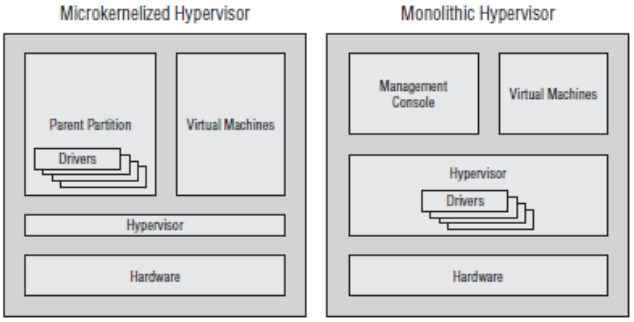

Fig 1: Micro-kernelized hypervisor vs. Monolithic hypervisor [11]

3. VIRTUALIZATION APPROACHES

The x86 architecture is the most popular computer architecture in enterprise datacenters today. Virtualization of the x86 architecture has been accomplished through two methods: full-virtualization or para-virtualization.

Full-virtualization is designed to provide total abstraction of the underlying physical system and creates a complete virtual system in which the guest operating systems can execute. No modification is required in the guest OS or application [12]; the guest OS or application is not aware of the virtualized environment so they have the capability to execute on the VM just as they would on a physical system [12]. This approach relies on binary translation to trap and virtualized the execution of certain sensitive, non-virtualizable instructions. With this approach, critical instructions are discovered (statically or dynamically at run-time) and replaced with traps into the VMM to be emulated in software. Binary translation can incur a large performance overhead in comparison to a virtual machine running on natively virtualized architectures [13].

In contrast, the para-virtualization approach modifies the guest operating system to eliminate the need for binary translation [14]. This typically involves replacing any privileged operations that will only run in ring 0 of the CPU with calls to the hypervisor (known as hypercalls). The hypervisor in turn performs the task on behalf of the guest kernels. As a result, the guest kernels are aware that they are executing on a VM allowing for near-native performance [15]. This approach typically limits support to open source operating systems such as Linux which may be freely altered and proprietary operating systems where the owners have agreed to make the necessary code modifications to target a specific hypervisor [15].

Since 2006, Intel and AMD have introduced new processor instructions (Intel VT and AMD's AMD-V) to assist virtualization software. The Hardware-assisted virtualization is a platform virtualization approach that enables efficient full-virtualization using help from hardware capabilities and reduces the need to para-virtualize guest operating systems [13].

Figure 2 shows the comparison between the virtualization approaches.

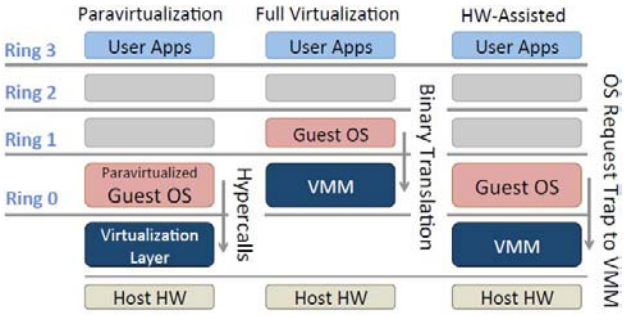

Fig 2: Comparison between the virtualization approaches [16]

4. HYPERVISORS ARCHITECTURES

Three hypervisors are used in our experiments: MS Hyper-V, VMware vSphere ESXi and Xen. Here is a brief overview on the architecture of each hypervisor:

©2009-2013 CIS Journal. All rights reserved.

4.1 Microsoft Hyper-V

Microsoft Hyper-V is a hypervisor-based virtualization technology for x64 versions of Windows Server [17]. It exists in two variants: as a stand-alone product called Hyper-V Server and as an installable role/component in Windows Server [18]. There is no difference between MS Hyper-V in each of these two variants. The hypervisor is the same regardless of the installed edition [18].

MS Hyper-V requires a processor with hardware-assisted virtualization functionality, enabling a much more compact virtualization codebase and associated performance improvements [10].

The Hyper-V architecture (figure 3) is based on micro-kernelized hypervisors. This is an approach where a host operating system, referred to as the parent partition, provides management features and the drivers for the hardware [11].

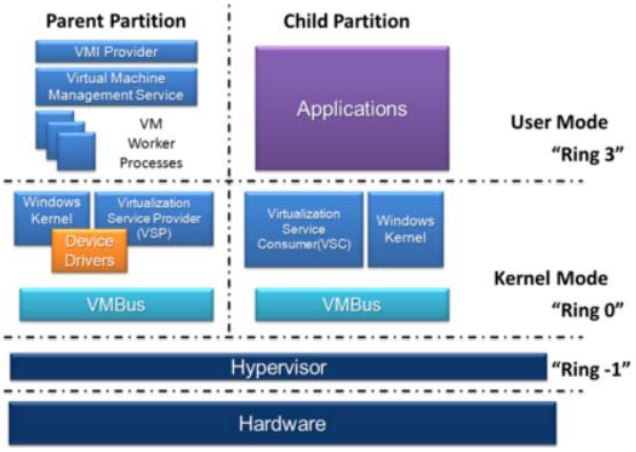

Fig 3: Hyper-V architecture [20]

MS Hyper-V implements isolation of virtual machines in terms of a partition (operating system and applications). A hypervisor instance has to have at least one parent partition, running a supported version of Windows Server [17]. The virtualization stack runs in the parent partition and has direct access to the hardware devices. The parent partition then creates the child partitions which host the guest OSs [17].

Child partitions do not have direct access to hardware resources. Hyper-V can host two categories of operating systems in the child partitions: Enlightened (the Microsoft naming for para-virtualization) and un-enlightened operating systems [19]. An enlightened partition has a virtual view of the resources, in terms of virtual devices. Any request to the virtual devices is redirected via the VMBus - a logical channel which enables inter-partition communication - to the devices in the parent partition managing the requests. Parent partitions run a Virtualization Service Provider (VSP), which connects to the VMBus and handles device access requests from child partitions [17]. The virtual devices of an enlightened child partition run a Virtualization Service Client (VSC), which redirect the request to VSPs in the parent partition via the VMBus [17]. The VSCs are drivers in the virtual machine, which together with other integration components are referred to as Enlightenments that provide advanced features and performance for a virtual machine. In contrast, the unenlightened child partition does not have the integration components and the VSCs; everything is emulated.

4.2 VMware ESXi

VMware vSphere is a software suite that has many software components such as vCenter, ESXi, and vSphere client [21]. VSphere is not a particular software that you can install and use, “it is just a package name which has other sub components” [21]. The core of the vSphere product suite is the hypervisor, which is the virtualization layer that serves as the foundation for the rest of the product line [22]. In vSphere 5, the hypervisor comes in the form of VMware ESXi which is a type 1 (bare-metal) hypervisor. All the virtual machines or Guest OS are installed on ESXi server. To install, manage and access those virtual servers which sit above of ESXi server, another part of vSphere suit called vSphere client or vCenter is needed [21]. In earlier versions of VMware vSphere, the hypervisor was available in two forms: VMware ESX and VMware ESXi [23]. Although both products shared the same core virtualization engine, supported the same set of virtualization features, leveraged the same licenses, and were both considered bare-metal installations, there were still notable architectural differences [23]. In ESX, VMware used a Linux-derived Service Console to provide an interactive environment through which users could interact with the hypervisor. The Linux-based Service Console also included services found in traditional operating systems, such as a firewall, Simple Network Management Protocol (SNMP) agents, and a web server [23].

VMware ESXi (figure 4), on the other hand, is the next generation of the VMware virtualization foundation. Unlike VMware ESX, ESXi installs and runs without the Linux-based Service Console. This gives ESXi an ultra light footprint of approximately 70 MB. Despite the lack of the Service Console, ESXi provides all the same virtualization features that VMware ESX supported in earlier versions [23].The key reason that VMware ESXi is able to support the same extensive set of virtualization functionality as VMware ESX without the Service Console is that the core of the virtualization functionality wasn’t (and still isn’t) found in the Service Console. It’s the VMkernel that is the foundation of the virtualization process. VMkernel manages the VMs’ access to the underlying physical hardware by providing CPU scheduling, memory management, and virtual switch data processing [22].

©2009-2013 CIS Journal. All rights reserved.

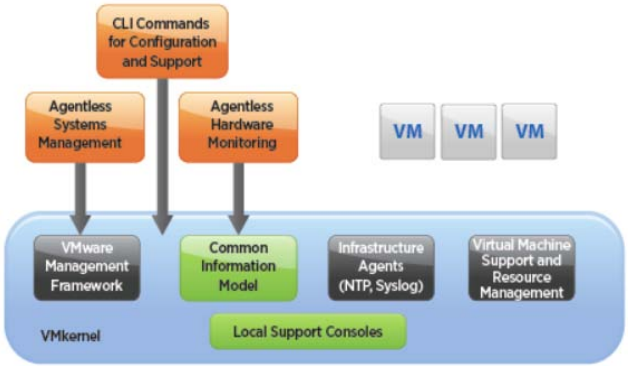

Fig 4: VMware ESXi architecture [24]

4.3 Xen

Xen [25] is a virtualization solution, originally developed at the University of Cambridge. It is the only bare-metal solution that is available as open source, and is used as the basis for a number of different commercial and open source applications [25].

Xen (figure 5) consists of several components that work together to deliver the virtualization environment. Its components are: Xen Hypervisor, Domain 0 Guest (referred as Dom0), and Domain U Guest (referred as DomU) which can be either Para-virtualized (PV) or Fully-Virtualized (FV)/Hardware-Assisted (HW-Assisted) Guest.

The Xen hypervisor is a software layer that runs directly on the hardware below any operating systems. It is responsible for CPU scheduling and memory partitioning of the various VMs running on the hardware device [26]. It is lightweight because it can delegate management of guest domains (DomU) to the privileged domain (Dom0) [27]. When Xen starts up, the Xen hypervisor takes first control of the system, and then loads the first guest OS, which is Dom0.

Dom0, a modified Linux kernel, is a unique virtual machine running on the Xen hypervisor that has special rights to access physical I/O resources as well as interact with the other virtual machines (DomU guests) [27].

DomU guests have no direct access to physical hardware on the machine as a Dom0 guest does and is often referred to as unprivileged. DomU PV guests are modified operating systems such as Linux, Solaris, FreeBSD, and other UNIX operating systems. DomU FV guests run standard Windows or any other unchanged operating system. [27]

Since Xen version 3.0, CREDIT [28] is the default Xen scheduler. It is designed to fairly share the CPUs among VMs. In the CREDIT scheduler, each VM is assigned a parameter called the weight; the CPU resources (or credit) are distributed to the virtual CPUs (vCPUs) of the VMs in proportion to their weight fairly. VCPUs are scheduled in a round-robin fashion. By default, when picking a vCPU, the scheduler allows it to run for 30ms. This quantum involves trade-offs between real-time performance and throughput [26].

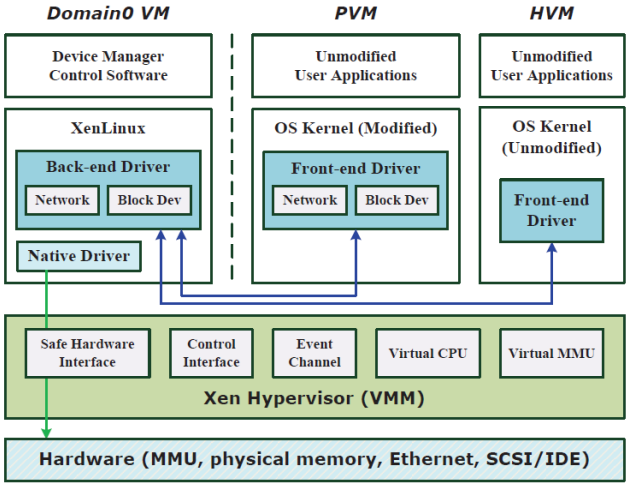

Fig 5: Xen architecture [29]

5. EXPERIMENTAL SETUP

5.1 Hardware Configuration

The hardware platform used for conducting the tests has the following characteristics: Intel® Desktop Board DH77KC, Intel® Xeon® Processor E3-1220v2 with four cores each running at a frequency of 3.1 GHz without hyper-threading support. The cache memory size is as follows: each core has 32 KB of L1 data cache, 32KB of L1 instruction cache and 256 KB of L2 cache. L3 cache is 8MB accessible to all cores.

5.2 Software Configuration

All the experiments in this paper are performed on MS Hyper-V Server 2012, VMware vSphere 5.1U1 ESXi, and Xen 4.2.1 (Opens use 12.3 is Dom-0).These versions were the latest shipping releases at the time of doing this evaluation (started in May 2013).

5.3 Guest operating System

Linux PREEMPT-RT v3.8.4-rt2 is the guest operating system used for our tests. Being an open source and configurable for usage in para-virtualized VM are the main reasons for selecting it as the guest OS. Both Linux versions (for para-virtualized and fully-virtualized VMs) are built using the build root [30] tool to make sure that the necessary modification are added to the kernel (Enlightenment drivers in case of Hyper-V and Xen drivers in case of Xen). The reason for choosing a real-time kernel is explained in the test metric below.

5.4 Virtual Machine Configuration

In case of Hyper-V server and Xen, two types of virtual machines are created: fully-virtualized and para-virtualized (named as Enlightened VM by Microsoft); only fully-virtualized VM is created for vSphere ESXi hypervisor.

As the used hardware has a new generation processor with hardware virtualization support, the

©2009-2013 CIS Journal. All rights reserved.

hardware-assisted virtualization (HW-Assisted) approach is used instead of the full-virtualization approach. So the naming of the created VMs are para-virtualized and HW-Assisted VMs.

However, we basically refer to the fact that in para-virtualization the kernel of the guest VM is modified, while in full and hardware virtualizations approaches the kernel of the guest is not modified.

Each VM is configured to have one virtual CPU and 1GB of memory.

The tests are done in each VM separately. Under Test VM (UTVM) is the name to be used for the VM (can be either para-virtualized or HW-Assisted) in which we conduct our tests.

6. TESTING PROCEDURES AND RESULTS

Although the test metrics explained below are mostly used to examine the real-time performance and behavior of real-time OSs on bare-machines [33] [34], they are useful to be used in other OS test cases. Moreover, virtualization together with real-time support emerges to be used in an increasing amount of use cases, varying from embedded systems to enterprise computing (e.g. RT-Xen). Therefore, these tests are a good fit for this paper evaluation.

6.1 Measuring Process

The Time Stamp Counter (TSC) is used for obtaining (tracing) the measurement values. It is a 64-bit register present on all x86 processors. The instruction RDTSC is used to return the TSC value. This counting register provides an excellent high-resolution, low-overhead way of getting CPU timing information and runs at a constant rate.

6.2 Testing Metrics

Below is an explanation of the evaluation tests. Note that the tests are initially done on a non-virtualized machine (further called Bare-Machine) as a reference, using the same OS as the UTVM.

6.2.1 Clock tick Processing Duration

This test examines the kernel clock tick processing duration in the guest OS (UTVM). Here is the test description: a real-time thread with the highest priority is created. This thread does a finite loop of the following tasks: starting the measurement by reading the time using RDTSC instruction, executing a "*busy loop*" that does some calculations and stopping the measurement by reading the time again using the same instruction. Having the time before and after the "*busy loop*" provides the time needed to finish its job. In case we run this test on the bare-machine, this "*busy loop*" will be delayed only by interrupt handlers. As we remove all other interrupt sources, only the clock tick timer interrupt can delay the "*busy loop*". When the "*busy loop*" is interrupted, its execution time increases.

When executing this test in a guest OS (VM) running on top of a hypervisor, it can be interrupted or scheduled away by the hypervisor as well, which will result in extra delays. To have accurate measurements we minimize the latency within the guest OS by using a real-time kernel variant.

Figure 6 presents the results of this test on the bare-machine, followed by an explanation. The X-axis indicates the time when a measurement sample is taken with reference to the start of the test. The Y-axis indicates the duration of the measured event; in this case the total duration of the "*busy loop*".

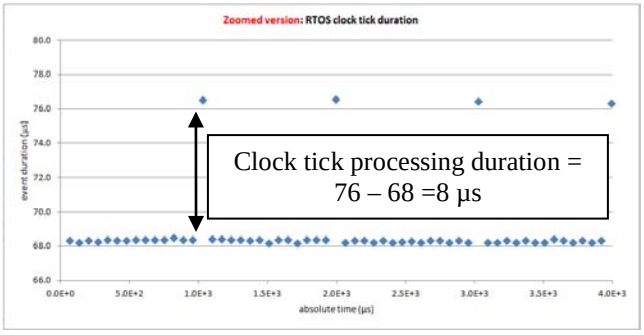

Fig 6: Clock tick processing duration of the bare-machine-zoomed

The lower values (68 µs) of figure 6 present the "busy loop" execution durations if no clock tick happens. In case of clock tick interruption, its execution is delayed until the clock interrupt is handled, which is 76 µs (top values). The difference between the two values is the delay spent handling the tick (executing the handler), which is 8 µs.

Note that the Linux kernel clock is configured to run at 1000 Hz, which corresponds to a tick each 1 ms. This is obvious in figure 6, which is a zoomed-in version of figure 7 below.

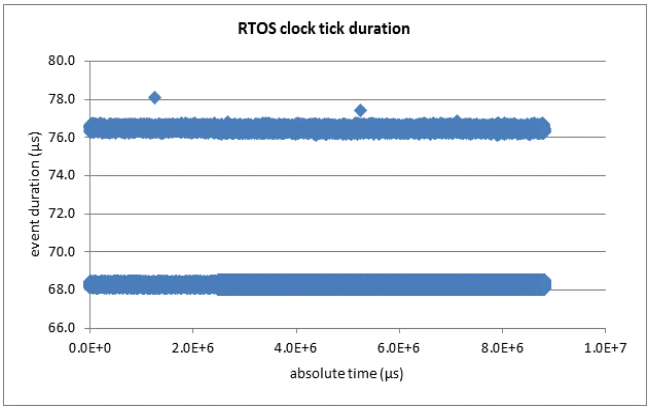

Fig 7: Clock tick processing duration of the bare-machine

Figure 7 represents the test results of 128000 captured samples, in a time frame of 9 seconds. Due to scaling reasons, the samples form a line.

©2009-2013 CIS Journal. All rights reserved.

As shown in figure 7, the "*busy loop*" execution time is 78 µs at some periods. Therefore, a clock tick delays any task by 8 to 10 µs.

This test detects all the delays that may occur in a system together with its behavior on the short term. To have a long-term view on the hypervisor behavior, we execute this test in the guest OS for a long duration (more than one hour) where 50 million samples are captured. The tables in the next sections show the maximum values obtained from the long term test.

6.2.2 Thread Switch Latency between Threads of Same Priority

This test measures the time needed to switch between threads having the same priority. Although real-time threads should be on different priority levels to be capable of applying rate monotonic scheduling theory [31], this test is executed with threads on the same priority level in order to easily measure thread switch latency without interference of something else.

For this test, threads must voluntarily yield the processor for other threads, so the SCHED_FIFO scheduling policy is used. If we didn't use the FIFO policy, a round-robin clock event could occur between the yield and the trace, and then the thread activation would not be seen in the test trace. The test looks for worst-case behaviour and therefore it is done with an increasing number of threads, starting with 2 and going up to 1000. As we increase the number of active threads, the caching effect becomes visible as the thread context will no longer be able to reside in the cache.

On the bare machine, this test is effected only by the clock tick interrupts, while in a VM on top of the hypervisor, it will be affected also by the scheduling and runtime behaviour of the hypervisor.

This test is used to depict the short time (less than a second) behaviour of the hypervisor. Therefore it is well suited to detect the normal scheduling behaviour of the hypervisor together with the impact caused by bus contention between virtual machines which is explained more in details later on.

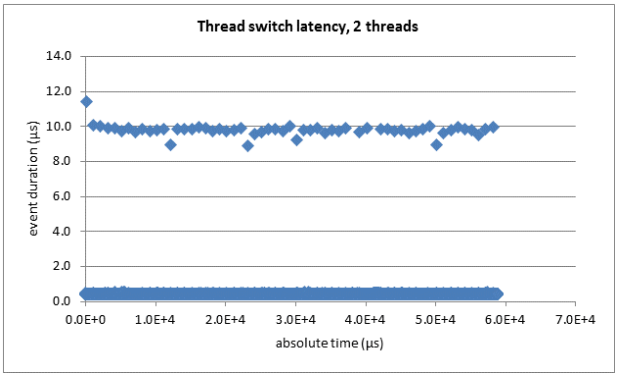

Fig 8: Thread switch latency between 2 threads on the Bare-machine

Figure 8 shows that the minimum switch latency between 2 threads is around 0.43 µs; the maximum latency is 11.45 µs which is dependent on the clock tick processing duration. Table 1 below shows the results of performing this test on the bare-machine using 2 and 1000 threads.

Table 1: Comparing the "Thread switch latency" results for the bare-machine

Test	Maximum
Switch latency between 2 threads	11.2 µs
Switch latency between 1000 threads	11.45 µs

Both tests, "Clock tick processing duration" and "Threads switch latency" are done on the para-virtualized and HW-Assisted VMs, using the several scenarios described in section C.

6.3 Testing Scenarios

This section is a description about the scenarios used for the evaluation.

6.3.1 Scenario 1: One-to-All

In this scenario, the UTVM (always with one vCPU) is the only running VM. In case of MS Hyper-V and Xen, there is also the parent partition (Dom-0) running but in idle state. The parent partition in Hyper-V runs by default on CPU0 [32] but can use any other CPU later on during runtime, while in Xen we manually configured it to run only on CPU0. The vCPU of the UTVM can run on any core (four physical CPUs available) during runtime. No affinity is used here. The aim of this scenario is to detect the pure hypervisor overhead (as there is no contention) together with its scheduling behavior (Does the UTVM vCPU runs all the time on the same CPU or shares the processing resources of all the available cores).

The figures below show the results of the "clock tick processing duration" test performed on each hypervisor.

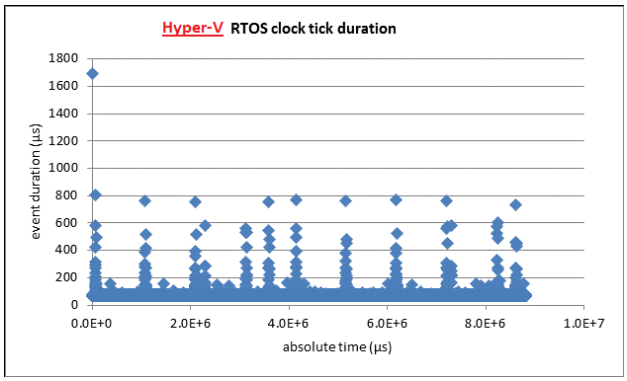

Fig 9: "Clock tick processing duration" test results on the Hyper-V para-virtualized (Enlightened) VM

©2009-2013 CIS Journal. All rights reserved.

Figure 9 shows the results of executing the "clock tick processing duration" test on a para-virtualized VM on top of Hyper-V. The HW-Assisted UTVM behaves exactly the same but with higher values. It is clear in figure 9 that every second (X-axis) the hypervisor is doing some tasks/scheduling decisions which causes the UTVM to be suspended/scheduled-away resulting in such high values periodically. Our policy is doing black-box testing which makes it difficult to understand the internal behavior of an out-of-the-box product.

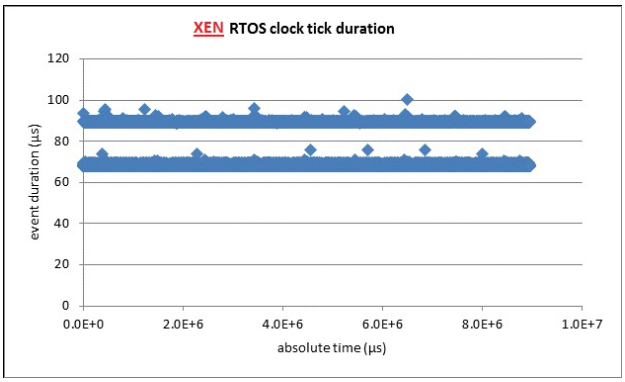

Fig 10: "Clock tick processing duration" test results on Xen HW-Assisted VM

Figure 10 shows the results of executing the "clock tick processing duration" test on a HW-Assisted VM using Xen hypervisor. This figure shows that the results are stable and predictable, which indicates that the Xen scheduler maps the UTVM vCPU to the same core all the time. No strange behavior is detected.

The para-virtualized UTVM on Xen has the same behavior but with lower values.

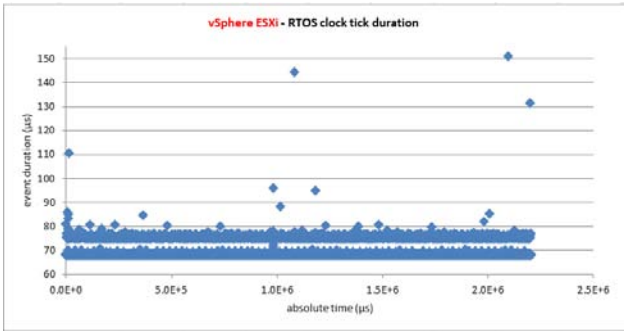

Fig 11: "Clock tick processing duration" test results on vSphere ESXi HW-Assisted VM

Figure 11 shows the results of executing the "clock tick processing duration" test on a HW-Assisted VM using vSphere ESXi hypervisor. The figure shows that the results are stable on the short term but unpredictable on the long term which can be seen in table 2; high measurements are detected at certain moments, which indicates that an unexpected behavior could happen during the system runtime.

The above figures show the behavior and performance of the hypervisors in a short time duration (mostly 10 seconds). The same test is done for a long period of 1 hour (= 50 million samples captured) several times on the three hypervisors to detect the worst-case duration.

Table 2: Comparison between the maximum "clock tick processing duration" in scenario 1

Maximum clock tick processing duration	
Bare-Machine	10 µs
Hyper-V HW-Assisted VM	4.35 ms
Hyper-V Para-virtualized VM	1.62 ms
Xen HW-Assisted VM	35 µs
Xen Para-virtualized VM	23 µs
ESXi HW-Assisted VM	59.61 ms

Table 2 shows a comparison between the worst case "clock tick processing duration" captured on each virtualization approach supported by the three hypervisors. The para-virtualization approach in both Hyper-V and Xen performs better than the corresponding HW-Assisted approach. Clearly, Xen para-virtualization performs the best among others.

The "Thread switch latency" test is also done on the virtualization approaches supported by each hypervisor. Table 3 shows the results of this test.

Table 3: Comparison between the maximum "Thread switch latency" results using 2 and 1000 threads, in scenario 1

Maximum thread switch latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Hyper-V HW-Assisted VM	63 µs	135 µs
Hyper-V Para-virtualized VM	45 µs	106 µs
Xen HW-Assisted VM	27.3 µs	27.4 µs
Xen Para-virtualized VM	15.27 µs	17.2 µs
ESXi HW-Assisted VM	19.28 µs	25.7 µs

Note that the results of this test are influenced by the processing durations of the clock tick and other interrupts that may occur in the system during the testing time, just like it is was detected during the long-term clock tick processing duration test. As this test duration is short, there will be a lot of random variations for each test run especially for Hyper-V and ESXi whose behavior is unpredictable.

This explanation applies to all the results in the next scenarios.

©2009-2013 CIS Journal. All rights reserved.

6.3.2 Scenario 2: One-to-One (Affinity)

This scenario is exactly the same as the previous one except that the UTVM vCPU is explicitly assigned to run always on one physical CPU using the affinity (pinning) configuration options of ESXi and Xen. Hyper-V does not support affinity, and therefore three CPUs are disabled from the BIOS. The aim of this scenario is to show the performance difference between Affinity and No-Affinity cases.

Tables 4 and 5 below show the tests' results of this scenario.

Table 4: Comparison between the maximum "clock tick processing duration" in scenario 2

Maximum clock tick processing duration	
Bare-Machine	10 µs
Hyper-V HW-Assisted VM	7.19 ms
Hyper-V Para-virtualized VM	4.08 ms
Xen HW-Assisted VM	35 µs
Xen Para-virtualized VM	23 µs
ESXi HW-Assisted VM	185.7 ms

Xen performs exactly the same as scenario 1 which shows that it runs the VM vCPU always on the same physical CPU in both affinity and no-affinity cases. For Hyper-V HW-Assisted VM, the results of this scenario are almost twice higher than the corresponding ones in scenario1, while it is more than twice for the para-virtualized VM. For ESXi, this increase is more than a factor of three.

Table 5 below shows the results of the "Thread switch latency" test.

Table 5: Comparison between the maximum "Thread switch latency" results using 2 and 1000 threads, in scenario 2

Maximum thread switch latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Hyper-V HW-Assisted VM	1.78 ms	1.8 ms
Hyper-V Para-virtualized VM	1.24 ms	1.47 ms
Xen HW-Assisted VM	27.3 µs	27.4 µs
Xen Para-virtualized VM	15.27 µs	17.2 µs
ESXi HW-Assisted VM	1.88 ms	2.11 ms

Again, the results of this test are dependent on the values obtained by the "clock tick processing duration".

6.3.3 Scenario 3: Contention with one CPU-Load VM

This scenario has two VMs: UTVM and CPU-Load VM which are both configured to run on the **same** physical CPU. The CPU-Load VM is running a CPU-stress program which is an infinite loop of mathematical calculations. The aim of this scenario is to explore the scheduling mechanism of the hypervisor between competing VMs.

As already mentioned before, Xen uses the Credit scheduler where each vCPU of a VM is scheduled to run for a quantum of 30 ms in a round-robin fashion. But, as Xen is an open source hypervisor, this quantum can be changed depending on the usage arena, with a minimum value of 1 ms. Therefore, in this scenario, we conducted the tests on Xen using the default and minimum scheduler quantum values (30 ms and 1 ms).

Table 6 shows the results of the "Clock tick processing duration test".

Table 6: Comparison between the maximum "clock tick processing duration" in scenario 3

Maximum clock tick processing duration	
Bare-Machine	10 µs
Hyper-V HW-Assisted VM	18.56 ms
Hyper-V Para-virtualized VM	9.16 ms
Xen HW-Assisted VM	30.16 ms 7.44 ms
Xen Para-virtualized VM	30.13ms 6.74 ms
ESXi HW-Assisted VM	281.71 ms

In table 6, there are two values for each Xen VM. For Xen HW-assisted VM, the worst case duration captured is 30.16 ms in case of using the default quantum (30 ms) and 7.44 ms in case of 1 ms quantum.

Xen outperforms the others if it is configured to use the minimum quantum value. Otherwise, Hyper-V para-virtualized VM performs the best.

Table 7 below shows the results of the "Thread switch latency" test.

©2009-2013 CIS Journal. All rights reserved.

Table 7: Comparison between the maximum "Thread switch latency" results using 2 and 1000 threads, in scenario 3

Maximum thread switch latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Hyper-V HW-Assisted VM	5.96 ms	5.96 ms
Hyper-V Para-virtualized VM	5.7 ms	5.17 ms
Xen HW-Assisted VM	30.08 ms 1.08 ms	30.1 ms 1.09 ms
Xen Para-virtualized VM	30.06 ms 1.06 ms	30.07 ms 1.07 ms
ESXi HW-Assisted VM	1.95 m	2.26 ms

Again, if Xen is configured to run using the minimum quantum, then Xen para-virtualized VM performs the best. Otherwise, ESXi is the best.

6.3.4 Scenario 4: Contention with one Memory-Load VM

This scenario is exactly the same as scenario 3 except using Memory-Load VM instead of CPU-Load VM. The Memory-Load VM runs an infinite loop of memcpy() function that copies 9 MB (a value that is larger than the whole caches) from one object to another. The goal of this scenario is to detect the cache effects on the performance of the UTVM as both are running on the same core and therefore sharing the same L1 and L2 caches.

Table 8 shows the results of the "clock tick processing duration test".

Table 8: Comparison between the maximum "clock tick processing duration" in scenario 4

Maximum clock tick processing duration	
Bare-Machine	10 µs
Hyper-V HW-Assisted VM	21.03 ms
Hyper-V Para-virtualized VM	11.8 ms
Xen HW-Assisted VM	30.18 ms 8.53 ms
Xen Para-virtualized VM	30.15 ms 7.3 ms
ESXi HW-Assisted VM	286.5 ms

The results of this scenario are in average 2ms greater than the ones from the previous scenario (scenario 3) which is due to the Memory-Load VM flushing the cache.

Table 9 shows the test results of the "Thread switch latency test".

Table 9: Comparison between the maximum "Thread switch latency" results using 2 and 1000 threads, in scenario 4

Maximum thread switch latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Hyper-V HW-Assisted VM	6.8 ms	6.86 ms
Hyper-V Para-virtualized VM	6.72 ms	5.24 ms
Xen HW-Assisted VM	30.1 ms 1.1 ms	30.11 ms 1.1 ms
Xen Para-virtualized VM	30.08 ms 1.08 ms	30.09 ms 1.09 ms
ESXi HW-Assisted VM	2.62 ms	2.8 ms

6.3.5 Scenario 5: All-to-All with 3 CPU-Load VMs

In this scenario, we run concurrently four VMs: the UTVM and three CPU-Load VMs. Theoretically, with such setup, each VM should run on a different physical CPU. The aim of this scenario is to confirm whether this expected behavior occurs.

If each VM runs on a separate CPU as expected, then the results of this scenario should be close to the ones of scenario 1. Table 10 shows the results of this scenario. Comparing the two scenarios shows that the performance difference is not big, which emphasize the good scheduling behavior/algorithm of each scheduler.

Table 10: Comparison between the maximum "clock tick processing duration" in scenario 5

Maximum clock tick processing duration	
Bare-Machine	10 µs
Hyper-V HW-Assisted VM	5.21 ms
Hyper-V Para-virtualized VM	2.55 ms
Xen HW-Assisted VM	35 µs
Xen Para-virtualized VM	23 µs
ESXi HW-Assisted VM	51.4 ms

Table 11 shows the results of the "thread switch latency" test. Again, Xen para-virtualized VM outperforms the rest.

Table 11: Comparison between the maximum "Thread switch latency" results using 2 and 1000 threads, in scenario 5

Maximum thread switch latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Hyper-V HW-Assisted VM	76 µs	165 µs
Hyper-V Para-virtualized VM	36 µs	108 µs
Xen HW-Assisted VM	27.3 µs	27.4 µs
Xen Para-virtualized VM	15.27 µs	17.2 µs
ESXi HW-Assisted VM	45.45 µs	124.8 µs

©2009-2013 CIS Journal. All rights reserved.

6.3.6 Scenario 6: All-to-All with three Memory-Load VMs

The setup of this scenario is exactly the same as the previous one (scenario 5) except using Memory-Load VMs instead of CPU-Load VMs.

Table 12: Comparison between the maximum "clock tick processing duration" in scenario 6

Maximum clock tick processing duration	
Bare-Machine	10 µs
Hyper-V HW-Assisted VM	15.18 ms
Hyper-V Para-virtualized VM	4.53 ms
Xen HW-Assisted VM	102 µs
Xen Para-virtualized VM	62 µs
ESXi HW-Assisted VM	70.79 ms

Table 12 shows the test results. Despite that the same number of VMs is used in this scenario and the previous one, the values measured here are in average twice greater than the ones from previous scenario. The reason for this difference is due to the "System bus bottleneck in SMP systems" concept. The hardware used in this evaluation is a Symmetric Multiprocessing (SMP) system with four identical processors connected to a single shared main memory using a "system bus". This "system memory bus" or "system bus" can be used by only one core at a time. If two processors are executing tasks that need to use the "system bus" at the same time, then one of them will use the bus while the other will be blocked for some time.

Referring back to the analysis of the performance difference between the two scenarios, scenario 5 is not causing high overheads because each CPU-Load VM is running a stress program that is quite small and fits in the core cache together with its data. Therefore, the three CPU-Loading VMs are not intensively loading the system bus which in turn will not highly affect the UTVM. While in scenario 6, the three Memory-Load VMs are intensively using the system bus. The UTVM is also running and needs to use the system bus from time to time. Therefore, the system bus is shared most of the time between four VMs (UTVM and 3 Memory-Load VMs), which causes extra contention. Thus, the more cores in the system that are accessing the system bus simultaneously, the more contention will occur and thus the overhead increases.

To explicitly show this effect, we created another additional scenario (scenario 7 below) where only one Memory-Load VM is running together with the UTVM.

Table 13 below shows the results of the "Thread switch latency" test. Xen, as always, performs the best.

Table 13: Comparison between the maximum "Thread switch latency" results using 2 and 1000 threads, in scenario 6

Maximum thread switch latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Hyper-V HW-Assisted VM	167 µs	445 µs
Hyper-V Para-virtualized VM	87 µs	393 µs
Xen HW-Assisted VM	66.5 µs	70 µs
Xen Para-virtualized VM	34 µs	40.4 µs
ESXi HW-Assisted VM	226.8 µs	283.1 µs

6.3.7 Scenario 7: Two-to-All with one Memory-Load VM

This scenario has two running VMs: the UTVM and a Memory-Load VM.

Table 14: Comparison between the maximum "clock tick processing duration" in scenario 7

Maximum clock tick processing duration	
Bare-Machine	10 µs
Hyper-V HW-Assisted VM	5.3 ms
Hyper-V Para-virtualized VM	2.12 ms
Xen HW-Assisted VM	49 µs
Xen Para-virtualized VM	32 µs
ESXi HW-Assisted VM	60.14 ms

The results of this scenario are a bit higher than the ones of scenario 1 which has only the UTVM running. This small difference is due to the effect of one Memory-Load VM running concurrently with the UTVM. This difference increases (as scenario 6) as the number of Memory-Load VMs increase.

Figure 15 shows the results of the "Thread switch latency" test.

Table 15: Comparison between the maximum "Thread switch latency" results using 2 and 1000 threads, in scenario 7

Maximum thread switch latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Hyper-V HW-Assisted VM	114 µs	212 µs
Hyper-V Para-virtualized VM	67 µs	166 µs
Xen HW-Assisted VM	42.7 µs	35.2 µs
Xen Para-virtualized VM	20.4 µs	21.4 µs
ESXi HW-Assisted VM	68 µs	212 µs

©2009-2013 CIS Journal. All rights reserved.

7. CONCLUSION

Nowadays, there are numerous virtualization platforms ranging from open-source hypervisors to commercial ones. These hypervisors can be either bare-metal running directly on top of the hardware or hosted being as an application in an operating system (OS). This paper provides a performance comparison between the top three leading bare-metal hypervisor families which are Microsoft (MS) Hyper-V, VMware ESXi and Xen.

MS Hyper-V and Xen are micro-kernelized hypervisors leveraging para-virtualization together with full-virtualization (hardware-assisted) while VMware ESXi is a monolithic hypervisor leveraging just full-virtualization approach (hardware-assisted).

The hardware used for this evaluation has a recent generation processor that supports the hardware-assisted (HW-Assisted) virtualization approach. Therefore, Para-virtualized and HW-Assisted VMs running Linux PREEMPT-RT as the guest OS are used. We conducted performance experiments using different scenarios on each virtualization approach supported by the latest versions of the mentioned hypervisors.

The para-virtualization approach in MS Hyper-V performs better than its corresponding HW-Assisted approach by a factor of two, while it is a factor of 1.5 for Xen. Moreover, the performance of Xen para-virtualization is much better than the one of MS Hyper-V. Comparing the results of the HW-Assisted approach supported by the three evaluated hypervisors, Xen outperforms the others while ESXi comes in the last position.

The performance of Xen (for both para-virtualization and HW-Assisted) is very close to the bare-machine (non-virtualized hardware) which makes Xen a candidate to be used for soft real-time OSs. This is due to the fact that Xen is an open source platform which allows the user to have full control of the whole system behavior (i.e.Dom-0 can be configured to run always on a specific processor without interfering other VMs running on other processors). MS Hyper-V and ESXi hypervisors do not adapt their scheduling policy in such flexible way. Also in case of contention, Xen offers the possibility to modify the quantum that a VM is scheduled-away which can reduce latencies in the VM.

In case real-time applications are concerned, running several concurrent VMs on a symmetric multiprocessor (SMP) system with a shared memory bus may increase latencies up to a factor of three. Utilizing at most two CPU's will use the memory bus in an interleaved mode. Once three or more CPU's are used, bus bottlenecks will emerge. This phenomena can be reduced by using large caches on different levels, however introducing a lot of uncertainty making these architectures only useful in business applications excluding real-time ones. This behavior applies to all the evaluated hypervisors as it is purely related with the shared hardware resources bottleneck.

REFERENCES

[1] Fatma Bazargan, Chan Yeob Yeun, Mohamed Jamal Zemerly, State-of-the-Art of Virtualization, its Security Threats and Deployment Models, international Journal for Information Security Research (IJISR),Volume 2, Issues 3/4, September/December 2012.

[2] Naveed Alam, Survey On Hypervisors, http://salsahpc.indiana.edu/b534projects/sites/default/files/public/6_Survey On Hypervisors_Alam Naveed Imran.pdf, 2009

[3] Ankita Desai, Rachana Oza, Pratik Sharma, Bhautik Patel, Hypervisor: A Survey on Concepts and Taxonomy, International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-2, Issue-3, February 2013

[4] VMWare, "Understanding Full virtualization, Para virtualization and hardware Assist," 2007. Available: http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf.

[5] EdwinYuen, How would explain the core differences in Hyper-V from VMware's offerings?, http://itknowledgeexchange.techtarget.com/itanswers/how-would-explain-the-core-differences-in-hyper-v-from-vmwares-offerings/, 2009

[6] Andy Syrewicze, VMware vs. Hyper-V: Architectural Differences,http://syrewiczeit.com/vmware-vs-hyper-v-architectural-differences/, 2013

[7] John Savill, What's the difference between monolithic and microkernelized hypervisors?, http://windowsitpro.com/virtualization/q-what-s-difference-between-monolithic-and-microkernelized-hypervisors, 2008

[8] Contel Bradford, Virtualization Wars: VMware vs. Hyper V: Which is Right For Your Virtual Environment?, http://www.storagecraft.com/blog/virtualization-wars-vmware-vs-hyper-v-which-is-right-for-your-virtual-environment/, 2013

[9] Nirmal Sharma, Hyper-V and VMware vSphere Architectures: Pros and Cons, http://www.serverwatch.com/server-tutorials/microsoft-hyper-v-and-vmware-vsphere-architectures-advantages-and-disadvantages.html, 2013

[10] Z. H. Shah, Windows Server 2012 Hyper-V: Deploying Hyper-V Enterprise Server Virtualization Platform, Packt Publishing, 2013.

©2009-2013 CIS Journal. All rights reserved.

[11] A. Finn and P. Lownds, Mastering Hyper-V Deployment, Wiley Publishing Inc.

[12] T. Abels, P. Dhawam and B. Chandrasekaran, "An overview of Xen Virtualization," [Online]. Available: http://www.dell.com/downloads/global/power/ps3q05-20050191-abels.pdf

[13] Wikipedia, Hardware-assisted virtualization, http://en.wikipedia.org/wiki/Hardware-assisted_virtualization, 2013

[14] VMware, A performance comparison of hypervisors, www.vmware.com/pdf/hypervisor_performance.pdf

[15] Virtuatopia, An Overview of Virtualization Techniques,http://www.virtuatopia.com/index.php/An_Overview_of_Virtualization_Techniques

[16] Jinho Hwang, Sai Zeng, Frederick y Wu, and Timothy Wood, \A Component-Based Performance Comparison of Four Hypervisors," 13th IFIP/IEEE International Symposium on Integrated Network Management (IM) Technical Session, 2013

[17] Microsoft, "Hyper-V Architecture," [Online].Available:http://msdn.microsoft.com/enus/library/cc768520%28v=bts.10%29.aspx

[18] M. T. Blogs, "Hyper-V: Microkernelized or Monolithic," [Online]. Available: http://blogs.technet.com/b/chenley/archive/2011/02/23/hyper-v-microkernelized-or-monolithic.aspx.

[19] G. Knuth, "Microsoft Windows Server 2008 – Hyper-V solution overview," 2008. [Online].Available: http://www.brianmadden.com/blogs/gabeknuth/archive/2008/03/11/microsoft-windows-server-2008-hyper-v-solution-overview.aspx.

[20] Wikipedia, Hyper-V, http://en.wikipedia.org/wiki/Hyper-V, 2013

[21] MustBeGeek, Difference between vSphere, ESXi and vCenter, http://www.mustbegeek.com/difference-between-vsphere-esxi-and-vcenter/

[22] Scott Lowe, Mastering VMware Vsphere 5, Publisher: Wiley India/Goels Computer Hut (2012)

[23] Windows Admins, Introduction of vSphere 5 and its components, http://winadmins.wordpress.com/page/29/, 2011

[24] Kenneth van Surksum, Paper: VMware ESXi 5.0 Operations Guide, http://virtualization.info/en/news/2011/10/paper-vmware-esxi-5-0-operations-guide.html, 2011

[25] Linux Foundation, "The Xen Project, the powerful open source industry standard for virtualization," [Online]. Available: http://www.xenproject.org/.

[26] Linux Foundation, "How Xen Works," [Online]. Available: http://www archive.xenproject.org/files/Marketing/HowDoesXenWork.pdf

[27] Hwanju K., Heeseung J., and Joonwon L. XHive: Efficient Cooperative Caching for Virtual Machines , IEEE Transactions on Computers, VOL. 60, NO. 1, 2011.

[28] Xen, "Credit scheduler" [Online]. Available: http://wiki.xen.org/wiki/Credit_Scheduler

[29] Peijie Yuz, Mingyuan Xiax, Qian Linx, Min Zhuz, Shang Gaoz, Zhengwei Qiz, Kai Chenx, Haibing Guanx, Real-time Enhancement for Xen hypervisor, IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2010

[30] Erik Andersen-The Buildroot developers, buildroot, http://buildroot.uclibc.org/

[31] M. H. Klein, T. Ralya, B. Pollak, R. Obenza and M. G. Harbour, A practitioner's Handbook for Real-Time Analysis, USA: Kumer Academic Publishers, 1994. ISBN 0-7923-9361-9.

[32] M. T. Wiki, "Hyper-V Concepts - vCPU (Virtual Processor)," [Online]. Available: http://social.technet.microsoft.com/wiki/contents/articles/1234.hyper-v-concepts-vcpu-virtualprocessor.aspx?wa=wsignin1.0.

[33] Luc Perneel, Hasan Fayyad-Kazan and Martin Timmerman, Android and Real-Time Applications: Take Care! Journal of Emerging Trends in Computing and Information Sciences, 2013

[34] Hasan Fayyad-Kazan, Luc Perneel and Martin Timmerman, Linux PREEMPT-RT vs. commercial RTOSs: how big is the performance gap? GSTF Journal of Computing, 2013