Global Journal of Computer Science and Technology
Hardware & Computation
Volume 13 Issue 2 Version 1.0 Year 2013
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

A Performance Comparison Between Enlightenment and Emulation in Microsoft Hyper-V

By Hasan Fayyad-Kazan, Luc Perneel & Martin Timmerman
Vrije Universiteit Brussel, Belgium

Abstract- Microsoft (MS) Hyper-V is a native hypervisor that enables platform virtualization on x86-64 systems. It is a micro-kernelized hypervisor where a host operating system provides the drivers for the hardware. This approach leverages MS Hyper-V to support enlightenments (the Microsoft name for Paravirtualization) in addition to the hardware emulation virtualization technique.

This paper provides a quantitative performance comparison, using different tests and scenarios, between enlightened and emulated Virtual Machines (VMs) hosted by MS Hyper-V server 2012. The experimental results show that MS enlightenments improve performance by a factor of more than two.

Keywords: *virtualization, hyper-v, enlightenments, emulation.*

GJCST-A Classification : *C.1.4*

Strictly as per the compliance and regulations of:

A Performance Comparison Between Enlightenment and Emulation in Microsoft Hyper-V

Hasan Fayyad-Kazan [α], Luc Perneel [σ] & Martin Timmerman [ρ]

Abstract- Microsoft (MS) Hyper-V is a native hypervisor that enables platform virtualization on x86-64 systems. It is a micro-kernelized hypervisor where a host operating system provides the drivers for the hardware. This approach leverages MS Hyper-V to support enlightenments (the Microsoft name for Paravirtualization) in addition to the hardware emulation virtualization technique.

This paper provides a quantitative performance comparison, using different tests and scenarios, between enlightened and emulated Virtual Machines (VMs) hosted by MS Hyper-V server 2012. The experimental results show that MS enlightenments improve performance by a factor of more than two.

Keywords: virtualization, hyper-v, enlightenments, emulation.

I. Introduction

Virtualization has become a popular way to make more efficient use of server resources within both private data centers and public cloud platforms. It refers to the creation of a Virtual Machine (VM) which acts as a real computer with an operating system (OS) [1]. It also allows sharing the underlying physical machine resources with different VMs.

The software layer providing the virtualization is called a Virtual Machine Monitor (VMM) or hypervisor [1]. It can be either Type 1 (or native, bare metal) running directly on the host's hardware to control the hardware and to manage guest operating systems, or Type 2 (or hosted) running within a conventional operating-system environment.

Since it has direct access to the hardware resources rather than going through an operating system, a native hypervisor is more efficient than a hosted architecture and delivers greater scalability, robustness and performance [2].

Microsoft Hyper-V implements Type 1 hypervisor virtualization [3]. In this approach, a hypervisor runs directly on the hardware of the host system and is responsible for sharing the physical hardware resources with multiple virtual machines [4]. In basic terms, the primary purpose of the hypervisor is to manage the physical CPU(s) and memory allocation between the various virtual machines running on the host system.

There are several ways to implement virtualization. Two leading approaches are Full virtualization (FV)/Hardware emulation and Para-virtualization (PV) [5]. Enlightenment is the Microsoft name for Para-virtualization.

This paper provides a quantitative performance comparison between hardware emulation and Enlightenments (Para-Virtualization) techniques hosted by MS Hyper-V server 2012.

It is organized as follows: Section 2 describes MS Hyper-V architecture and Enlightenment approach; Section 3 shows the experimental setup used for our evaluation; Section 4 explains the test metrics, scenarios and results obtained; and section 5 gives a final conclusion.

II. Microsoft Hyper-V

Microsoft Hyper-V is a hypervisor-based virtualization technology for x64 versions of Windows Server [6]. It exists in two variants: as a stand-alone product called Hyper-V Server and as an installable role/component in Windows Server [7].

There is no difference between MS Hyper-V in each of these two variants. The hypervisor is the same regardless of the installed edition [7].

MS Hyper-V requires a processor with hardwareassisted virtualization functionality, enabling a much more compact virtualization codebase and associated performance improvements [3].

The Hyper-V architecture is based on microkernelized hypervisors (figure 1). This is an approach where a host operating system, referred to as the parent partition, provides management features and the drivers for the hardware [8].

Author α: PhD Candidate, Department of Electronics and Informatics, Vrije Universiteit Brussel, Pleinlaan 2- 1050 Brussels, Belgium.
e-mail: hafayyad@vub.ac.be
Author σ: PhD Candidate, Department of Electronics and Informatics, Vrije Universiteit Brussel, Pleinlaan 2- 1050 Brussels, Belgium.
e-mail: luc.perneel@vub.ac.be
Author ρ: Professor, Department of Electronics and Informatics,Vrije Universiteit Brussel, Pleinlaan 2- 1050 Brussels, Belgium.
e-mail: martin.timmerman@vub.ac.be

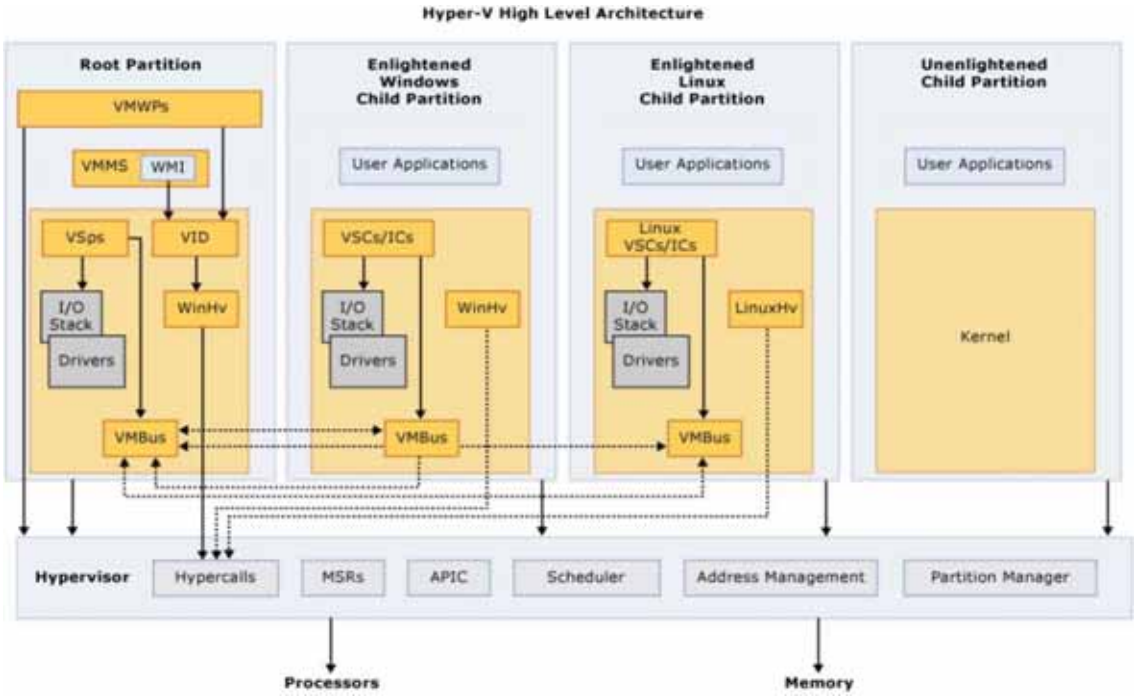


Figure 1 : Hyper-V Architecture [6]

With this approach, the only layer between a guest operating system and the hardware is a streamlined hypervisor with simple partitioning functionality. The hypervisor has no third-party device drivers [9]. The drivers required for hardware sharing reside in the host operating system, which provides access to the rich set of drivers already built for Windows [9].

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 [6].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 [6].

Child partitions do not have direct access to hardware resources. Hyper-V can host two categories of operating systems in the child partitions: Enlightened (Hyper-V Aware) and un-enlightened (Hyper-V Unaware) operating systems [10]. 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 (figure 1) – 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 [6]. Enlightened child partition virtual devices internally run a Virtualization Service Client (VSC) (figure1), which redirect the request to VSPs in the parent partition via the VMBus [6]. The VSCs are the drivers of 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.

III. Experimental Setup

Microsoft Hyper-V Server 2012 is tested here. It is a dedicated stand-alone product that contains the hypervisor, Windows Server driver model, virtualization capabilities, and supporting components such as failover clustering, but does not contain the robust set of features and roles found in the Windows Server operating system [11].

As MS Hyper-V supports enlightened and emulated VMs, both VMs are created, running Linux PREEMPTRT v3.8.4-rt2 [12]. Being open source and configurable for usage in enlightened VM are the main reasons for selecting it as the guest OS. This also permits us to compare with another ongoing study of XEN.

Both Linux versions (for enlightened and emulated VM) are built using the buildroot [13] tool to make sure that the enlightenment drivers are added to the enlightened VM.

The tests are done in each VM separately. Under Test VM (UTVM) is the name used for the tested VM, which can be either enlightened or emulated. Each VM has one virtual CPU (vCPU).

One physical CPU is allocated for each VM, using the "virtual machine reserve" and "virtual machine limit" attributes in the VM settings using Hyper-V Manager.

The hardware platform used for conducting the tests has the following characteristics: Intel® Desktop Board DH77KC, Intel® Xeon® Processor E3-1220v2 with 4 cores each running at a frequency of 3.1 GHz, and no 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. The system memory is 8 GB.

IV. Testing Procedures and Results

a) *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 since the Pentium. 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.

b) *Testing Metrics*

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

i. *Clock tick processing duration*

The kernel clock tick processing duration is examined here. The results of this test are extremely important as the clock interrupt - being on a high level interrupt on the used hardware platform - will bias all other performed measurements. Using a tickless kernel will not prevent this from happening as it will only lower the number of occurrences. The kernel is not using the tickless timer option.

Here is a description of how this test is performed: 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.

Running the same test in a VM also shows when it is scheduled away by the VMM, which in turn impacts latency.

Figure 2 presents the results of this test on the baremachine, 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 Yaxis indicates the duration of the measured event; in this case the total duration of the "busy loop".

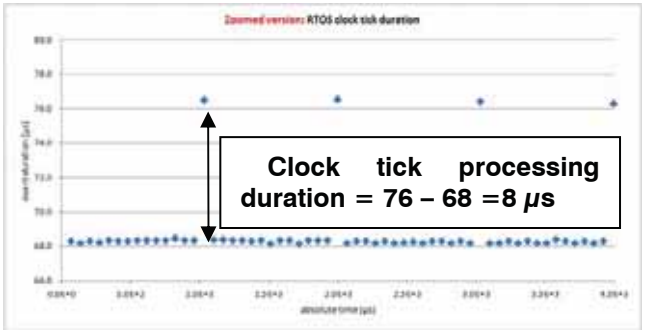


Figure 2 : Clock tick processing duration of the baremachine-zoomed

The lower values (68 μs) of figure 2 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 kernel clock is configured to run at 1000 Hz, which corresponds to a tick each 1 ms. This is obvious in figure 2, which is a zoomed version of figure 3 below.

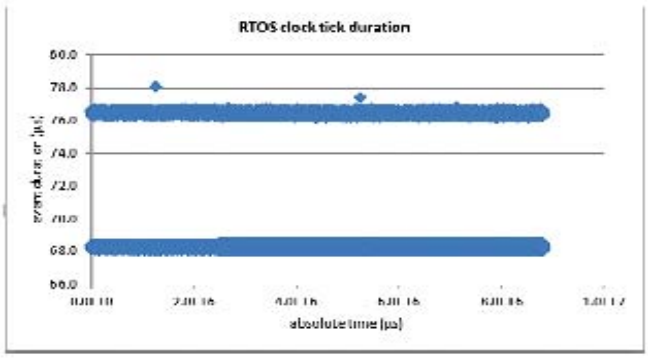


Figure 3 : Clock tick processing duration of the baremachine

Figure 3 represents the test results of 128000 captured samples, in a time frame of 9 seconds. Due to scaling reasons, the samples form a line. As shown in figure 3, 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 is very useful as it detects all the delays that may occur in a system during runtime. Therefore, we execute this test for long duration (more than one hour) to capture 50 million samples. The results in the tables of section D are comparing the maximum results obtained from the 50 million samples.

ii. *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 [16], 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 behavior 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.

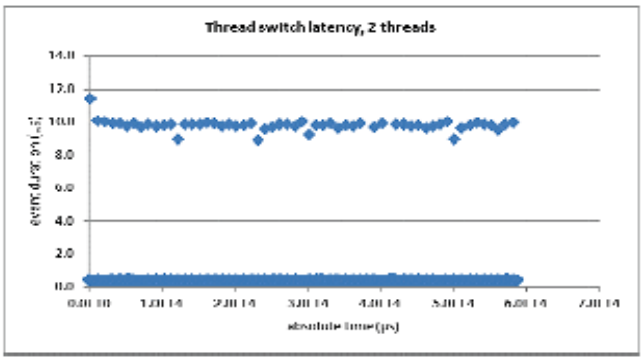


Figure 4 : Thread switch latency between 2 threads on the Bare-machine

Figure 4 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 enlightened and emulated VMs, using the several scenarios described in section D.

c) Processor Affinity in MS Hyper-V

Most virtualization solutions like Xen and VMware support the affinity concept where a vCPU of a VM can be tied to a given physical processor. Benjamin Armstrong, Hyper-V Program Manager, explains in the blog "Processor Affinity and why you do not need it on Hyper-V" [14] that there is no need for this concept in Hyper-V. Instead, one can reserve a physical CPU (pCPU) for the VM to guarantee that it always has a whole processor.

Moreover, if a VM has one vCPU and the host has more than one core, this VM can be mapped to any of the available cores in a round-robin way between all the cores [15]. The parent partition is the only VM parked on core 0.

d) Testing scenarios

Below is a description of the scenarios used for the evaluation. In all the scenarios drawings, the parent partition (VM) is not shown because it is idle.

i. *Scenario 1: One-to-All*

As shown in figure 5, this scenario has only one VM, the UTVM with one vCPU. This vCPU can run on any core (physical CPU) during runtime. The aim of this scenario is to detect the pure hypervisor overhead (as there is no contention).

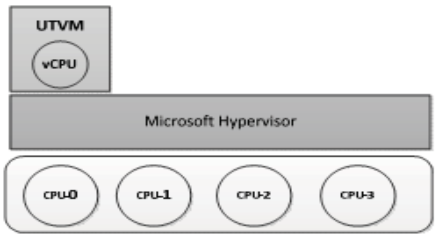


Figure 5 : One-to-All scenario

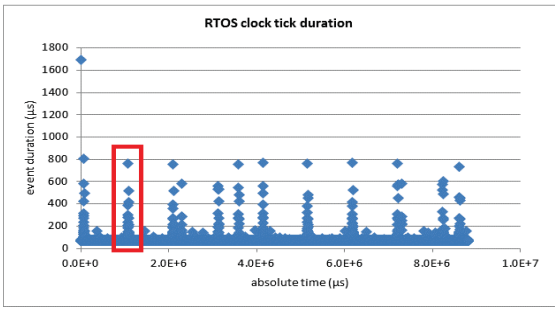


Figure 6 : Clock tick duration test for Enlightened VM in scenario 1

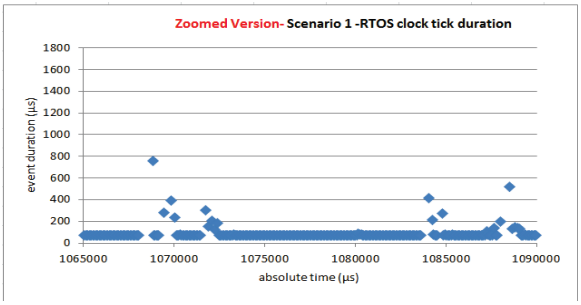


Figure 7 : Zoomed version of the red sport in figure 6

Figure 6 shows that every second, the hypervisor is doing some tasks/scheduling decisions which causes the VM to be suspended/scheduled-away resulting in such high values periodically.

Note that our policy is black-box testing which makes it difficult to understand the internal behavior of an out-of-the-box product.

The emulated VM behaves exactly the same except with higher values. Table 2 is a comparison between the "clock tick processing duration" test results for both VMs, while table 3 is a comparison for the "thread switch latency" test results.

Table 2 : Comparison between the "clock-tick processing duration" test results

Clock Tick Processing Duration	
	Maximum Overhead
Bare-Machine	10 µs
Emulated VM	4.35 ms
Enlightened VM	1.62 ms

Table 3 : Comparison between the "Thread switch latency" test results

Maximum Switch Latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Emulated VM	3.53 ms	1.24 ms
Enlightened VM	63 µs	687 µs

Note that the "maximum switch latency" in all the scenarios depends on the processing durations of clock tick and other interrupts that may occur in the system during the testing time.

ii. *Scenario2: One-to-One*

As mentioned before, Hyper-V does not support affinity. It sends the workload of a VM to the first physical CPU that is available.

In this scenario, there is only one physical CPU available, while the other three are disabled from the BIOS. There is only the UTVM, together with the parent partition which is always parked on CPU-0 but idle. Therefore, UTVM is also pinned to CPU-0.The aim of this scenario is to clarify if the affinity technique removes the periodic high measurements.

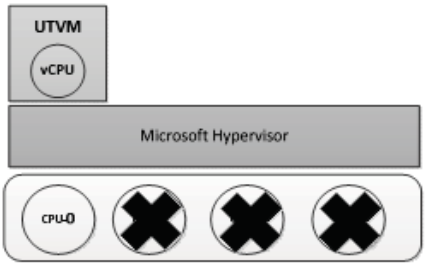


Figure 8 : One-to-One scenario

Figure 9 shows that the periodic high values are still detected.

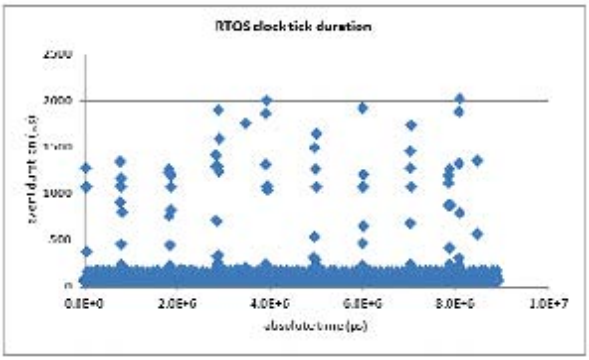


Figure 9 : Clock tick duration test for Enlightened VM in scenario 2

Tables 4 and 5 compares the results of the two tests: "Clock Tick processing duration" and "Thread switch latency".

Table 4 : Comparison between the "clock-tick processing duration" test results

Clock Tick Processing Duration	
	Maximum Overhead
Bare-Machine	10 µs
Emulated VM	7.19 ms
Enlightened VM	4.08 ms

Table 5 : Comparison between the "Thread switch latency" test results

Maximum Switch Latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Emulated VM	1.78 ms	1.8 ms
Enlightened VM	1.24 ms	1.47 ms

iii. *Scenario3: Contention with 1 CPU-Load VM*

This scenario has 2 VMs, UTVM and CPU-Load VM, both running on the same physical CPU as shown in figure 10. 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.

Each of the two VMs has its "virtual machine reserve" and "virtual machine limit" attributes set to 50%.

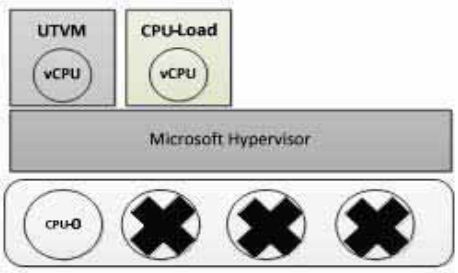


Figure 10 : Contention with 1 CPU-Load VM scenario

Tables 6 and 7 compares the results of the two tests: “Clock Tick processing duration” and “Thread switch latency”.

Table 6 : Comparison between the “clock-tick processing duration” test results

Clock Tick Processing Duration	
	Maximum Overhead
Bare-Machine	10 µs
Emulated VM	18.56 ms
Enlightened VM	9.16 ms

Table 7 : Comparison between the “Thread switch latency” test results

Maximum Switch Latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Emulated VM	5.96 ms	5.96 ms
Enlightened VM	5.7 ms	5.17 ms

iv. *Scenario4: Contention with 1 Memory-Load VM*

This scenario is exactly the same as scenario 3 except using Memory-Load VM instead of CPU-Load VM. This VM is running 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 other goal of this scenario using such a VM is to detect the cache effects on the performance of the UTVM.

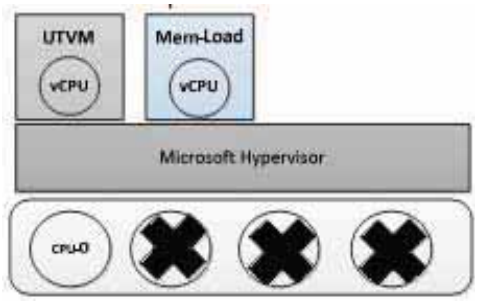


Figure 11 : Contention with 1 Memory-Load VM scenario

Tables 8 and 9 compare the results of the two tests: “Clock Tick processing duration” and “Thread switch latency”.

Table 8 : Comparison between the “clock-tick processing duration” test results

Clock Tick Processing Duration	
	Maximum Overhead
Bare-Machine	10 µs
Emulated VM	21.3 ms
Enlightened VM	11.8 ms

Table 8 shows that measurements of this scenario are greater than the ones of the previous scenario (scenario 3) by almost 3 ms.

Table 9 : Comparison between the “Thread switch latency” test results

Maximum Switch Latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Emulated VM	6.8 ms	6.8 ms
Enlightened VM	6.7 ms	5.2 ms

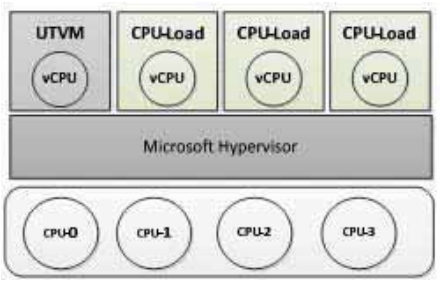


Figure 12 : All-to-All with 3 CPU-Load VMs scenario

Again, the periodic peaks are captured as shown in figure 13 below.

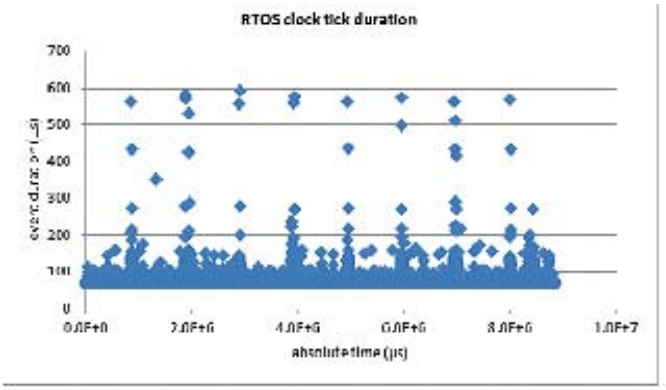


Figure 13 : Clock tick duration test for Enlightened VM in scenario 5

Tables 10 and 11 compare the results of the two tests: “Clock Tick processing duration” and “Thread switch latency”.

Table 10 : Comparison between the “clock-tick processing duration” test results

Clock Tick Processing Duration	
	Maximum Overhead
Bare-Machine	10 µs
Emulated VM	5.21 ms
Enlightened VM	2.55 ms

Table 11 : Comparison between the “Thread switch latency” test results

Maximum Switch Latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Emulated VM	76 µs	165 µs
Enlightened VM	36 µs	108 µs

vi. *Scenario6: All-to-All with 3 Memory-Load VMs*

This scenario is exactly the same as the previous scenario (scenario 5) except using Memory-Load VMs instead of CPU-Load VMs as shown in figure 14.

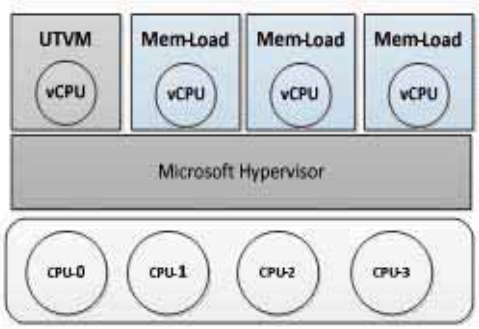


Figure 14 : All-to-All with 3 Memory-Load VMs scenario

Tables 12 and 13 compare the results of the two tests: "Clock Tick processing duration" and "Thread switch latency".

Table 12 : Comparison between the "clock-tick processing duration" test results

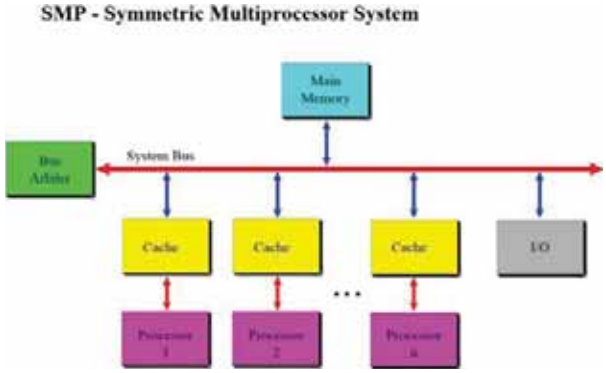
Clock Tick Processing Duration	
	Maximum Overhead
Bare-Machine	10 µs
Emulated VM	15.18 ms
Enlightened VM	4.53 ms

Table 13 : Comparison between the "Thread switch latency" test results

Maximum Switch Latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Emulated VM	167 µs	445 µs
Enlightened VM	87 µs	393 µs

The resulting values of this scenario are around three times greater than the ones of the previous scenario (scenario 5) even though the same number of VMs is running. This difference in the results is due to the concept explained in the following section (System bus bottleneck in SMP systems).

System bus bottleneck in SMP systems.



The hardware platform used for this evaluation is a Symmetric Multiprocessing (SMP) system with four identical processors connected to a single shared main memory using a system bus. They have full access to all I/O devices and are treatedequally.

The 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. As the processor used has 4 cores, when all of these are running at the same time, system bus contention occurs.

Scenario 5 is not causing high overheads because the CPU stress program 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.

Referring back to scenario 6, the three Memory-Load VMs are intensively using the system bus. The UTVM is also running and requires the usage of 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 sharing the resources with the UTVM. The following demonstrates our observation.

vii. *Scenario7: TWO-to-ALL with 1 Memory-Load VM*

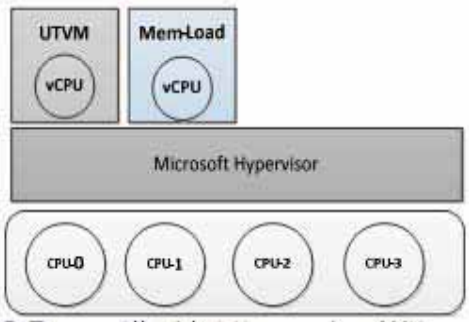


Figure 15 : Two-to-All with 1 Memory-Load VM scenario

Tables 15 and 16 compare the results of the two tests: "Clock Tick processing duration" and "Thread switch latency".

Clock Tick Processing Duration	
	Maximum Overhead
Bare-Machine	10 µs
Emulated VM	5.3 ms
Enlightened VM	2.12 ms

Table 15 : Comparison between the "clock-tick processing duration" test results

Maximum Switch Latency between:		
	2 Threads	1000 Threads
Bare-Machine	11.2 µs	11.45 µs
Emulated VM	114 µs	212 µs
Enlightened VM	67 µs	166 µs

Table 16 : Comparison between the "Thread switch latency" test results

V. Conclusion

Microsoft recent Hyper-V technology is a "Microkernalized Type 1" hypervisor which leverages paravirtualization (called Enlightenment by Microsoft) in addition to the traditional hardware emulation technique. It exists in two variants: as a stand-alone product called Hyper-V Server and as an installable role in Windows Server.

There is no difference between MS Hyper-V in each of these two variants. The hypervisor is the same regardless of the installed edition.

In this paper, MS Hyper-V Server 2012 is undergoing testing. It installs a very minimal set of Windows Server components to optimize the virtualization environment. It supports Enlightened (special drivers are added to the VM to provide advanced features and performance) and hardware-emulated VMs.

This work compares the performance between the two types of VM. For this purpose, different tests and several scenarios are used. The results show that the enlightened VM performs on average twice as good as the hardware-emulated VM. This performance enhancement may increase/decrease depending on the scenario in question.

Even with this improvement, Enlightened VM performance is low compared with bare-machine (non-virtualized) performance.

A shared-memory symmetric multiprocessor hardware with four physical cores is used for conducting the tests. The results also show that the synchronous usage of all the available cores causes an intensive overload in the system bus which in turn increases latencies by a factor of 3 when compared with a system with only one active core.

References Références Referencias

1. M. Lee, A. S. krishnakumar, P. Krishnan, S. Nayjot and Y. Shalini, "Supporting Sofy Real-Time Tasks in the Xen Hypervisor," in the 6th ACM SIGPLAN/SIGOPS international conference on Virtual execution enviroments, 2010.
2. VMWare, "Understanding Full virtualization, Paravirtualization and hardware Assist," 2007. [Online]. Available: http://www.vmware.com/files/pdf/VMware_pa ravirtualization.pdf.
3. Z. H. Shah, Windows Server 2012 Hyper-V: Deploying Hyper-V Enterprise Server Virtualization Platform, Packt Publishing, 2013.
4. Virtuatopia, "An Overview of the Hyper-V Architecture," [Online]. Available:http://www.virtuatopia.com/index.php/An_Overview_of_the_HyperV_Architecture.
5. T. Abels, P. Dhawam and B. Chandrasekaran, "An overview of Xen Virtualization," [Online]. Available: http://www.dell.com/downloads/global/power/ps3q05-2 0050191-abels.pdf.
6. Microsoft, "Hyper-V Architecture," [Online]. Available: http://msdn.microsoft.com/enus/library/cc768520%28v=bts.10%29.aspx.
7. 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.
8. Finn and P. Lownds, Mastering Hyper-V Deployment, Wiley Publishing Inc.
9. M. Corporation, "Windows Server 2008 Hyper-V Technical Overview," [Online]. Available: http://download.microsoft.com.
10. 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.
11. Microsoft, "Microsoft Hyper-V Server 2012," [Online]. Available: http://www.microsoft.com/enus/server-cloud/hyper-v-server/.
12. "CONFIG PREEMPT RT Patch-RT wiki," [Online]. Available:https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch.
13. T. B. developers, "Buildroot: Making Embedded Linux easy," [Online]. Available: http://buildroot.uclibc.org/.
14. B. Armstrong, "Hyper-V CPU Scheduling-Part 1," 2011. [Online]. Available: http://blogs.msdn.com/b/virtual_pc_guy/archive/2011/02/14/hyper-v-cpu-scheduling-part-1.aspx.
15. M. T. Wiki, "Hyper-V Concepts - vCPU (Virtual Processor)," [Online]. Available: http://social.technet.microsoft.com/wiki/contents/articles/1234.hyper-v-conceptsvcpuvirtualprocessor.aspx?wa=wsignin1.0.
16. 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.

Global Journals Inc. (US) Guidelines Handbook 2013

www.GlobalJournals.org