Business Hypervisors for Real-time Applications

Luc Perneel
Department of Electronics and Informatics, Vrije Universiteit Brussel, Belgium
luc.perneel@vub.ac.be

H. Fayyad-Kazan
Department of Electronics and Informatics, Vrije Universiteit Brussel, Belgium
hafayyad@vub.ac.be

Long Peng
Department of Electronics and Informatics, Vrije Universiteit Brussel and Dedicated-Systems Experts, Belgium
longpeng@vub.ac.be

Fei Guan
Department of Electronics and Informatics, Vrije Universiteit Brussel, Belgium
feiguan@vub.ac.be

Martin Timmerman
Department of Electronics and Informatics, Vrije Universiteit Brussel, Belgium and Dedicated-Systems Experts, Belgium
martin.timmerman@vub.ac.be

Abstract*— System virtualization is one of the hottest trends in information technology today. It is not just another nice to use technology but has become fundamental across the business world. It is successfully used with many business application classes where cloud computing is the most visual one. Recently, it started to be used for soft Real-Time (RT) applications such as IP telephony, media servers, audio and video streaming servers, automotive and communication systems in general. Running these applications on a traditional system (Hardware + Operating System) guarantee their Quality of Service (QoS); virtualizing them means inserting a new layer between the hardware and the (virtual) Operating System (OS), and thus adding extra overhead. Although these applications' areas do not always demand hard time guarantees, they require the underlying virtualization layer supports low latency and provide adequate computational resources for completion within a reasonable or predictable timeframe. These aspects are intimately intertwined with the logic of the hypervisor scheduler. In this paper, a series of tests are conducted on three hypervisors (VMware ESXi, Hyper-V server and Xen) to provide a benchmark of the latencies added to the applications running on top of them. These tests are conducted for different scenarios (use cases) to take into consideration all the parameters and configurations of the hypervisors' schedulers. Finally, this benchmark can be used as a reference for choosing the best hypervisor-application combination.

Keywords: ESXi; Hyper-V; Virtualization; Xen; Real-time

I. INTRODUCTION

System virtualization is a technology that allows the physical machine resources to be shared among different Virtual Machines (VMs) via the use of a software layer called hypervisor or Virtual Machine Monitor (VMM). The hypervisor can be either Type-1 which runs directly on the system hardware and thus is often referred to as bare-metal hypervisor, or Type-2 which runs as an application on top of a conventional Operating System (OS) and is referred to as hosted hypervisor [1, 2].

Due to the low cost availability of multi-core chips used in a symmetrical multiprocessor way, virtualization in the server and desktop world has already matured with both software and hardware solutions available for several years [3]. Currently, there are numerous virtualization products ranging from open-source hypervisors such as Xen, KVM, and OpenVZ to commercial ones as VMware vSphere ESXi, VirtualBox, OKL4, Parallels Workstation and Microsoft Hyper-V [3].

Theoretically speaking, as a bare-metal hypervisor has direct access to the hardware resources rather than going through a hosting operating system, it is more efficient, delivers greater scalability, robustness and performance [4]. Therefore, bare-metal hypervisors are our focus in this paper.

Virtualization is successfully used with many business application classes where cloud computing is the most visual one. However, the growing set of applications also includes soft-real time applications such as media-based ones. It is obvious that the hypervisor layer between the hardware and the OS running in a VM incurs extra overhead compared to a non-virtualized system. This overhead can vary from one hypervisor type to another. Moreover, there is no quantitative benchmark of these overheads (added latencies), which in turn makes it difficult for users to choose the hypervisor that fits their requirements.

Currently, there are just some technical publications [5, 6], initiated by the hypervisor vendors themselves, comparing mainly the hypervisors' supported features. More recently, a number of unbiased scientific papers originated by researchers focused on the I/O performance, network throughput and CPU utilization, using public benchmark tools which are not really adapted for the comparison job [7, 8]. As a consequence, most publications are not at all quantifying their statements; others give only a limited scientific proof.

Our contribution started from this point. We want to go beyond theoretical statements and quantify the performance differences (in matters of latencies) between virtualization solutions by focusing on their architectures and internal components such as scheduling policies.

To achieve this, different types of tests (explained further on) are executed on the top three market hypervisors: Microsoft (MS) Hyper-V Server 2012R2, Xen 4.2.1 and VMware VSphere ESXi 5.1U1. All these versions were the latest shipping releases at the time of doing this study. The tests are conducted for different use cases to take into consideration all the parameters and configurations of the hypervisors' schedulers.

II. HYPERVISORS ARCHITECTURES

A. Microsoft Hyper-V Server 2012 R2

Microsoft Hyper-V is a bare-metal micro-kernerlized hypervisor that enables platform virtualization on x86-64 systems [9]. It exists in two variants: as a stand-alone product called Hyper-V Server [10] and as an installable feature in Windows Server OS. In both versions, the hypervisor is exactly the same and must have at least one parent, or root VM, running Windows OS 64-bit Edition [11]. The difference between the two products are the features and components found in the parent (privileged) VM guest OS.

Once Hyper-V is installed, VMs atop it are created. All the VMs except the parent VM are called child VMs (un-privileged). Hyper-V supports a number of different types of guest OSs running in the child VMs. They can be either Hyper-V Aware (enlightened/para-virtualized) Windows OSs, Hyper-V Aware non-Windows OSs and Non Hyper-V Aware OSs [12]. The hypervisor needs to deal with CPU scheduling. The CPU scheduler shares the same role as conventional operating systems. This role is to assign execution contexts to processors in a way that meets system objectives such as responsiveness, throughput, and utilization. On conventional operating systems, the execution context corresponds to a process or a thread; on hypervisor, it corresponds to a VM.

When a VM is created atop Hyper-V, several configurations have to be set to define its behaviour when accessing hardware resources. The basic ones are shown in Figure 1 below followed by their explanations.

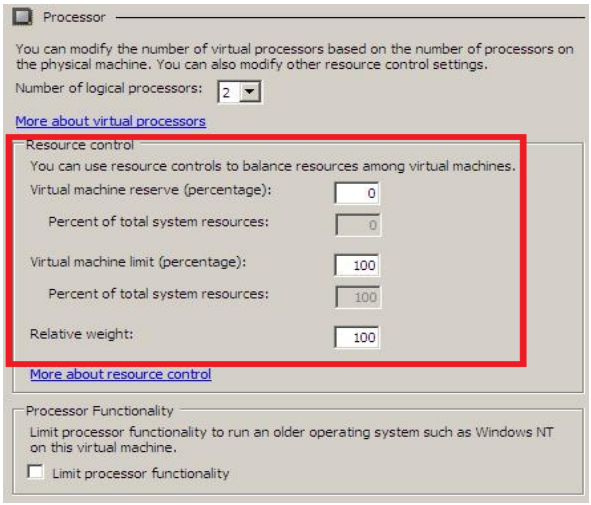

Fig. 1. Hyper-V VM configuration for resources access

Virtual machine reserve: This setting specifies the percentage of the physical machine's overall CPU resources reserved for this particular VM. This value can range from 1-100% and is relative to the number of processors assigned to the virtual machine (a VM can have more than one virtual processor). For instance, if a physical machine has four cores and you set the VM reserve value to 100 %, then the equivalent of an entire processor core will be reserved for a VM. If the VM sits idle at 10% most of the time, the other 90% of the processing time is still unavailable to any other VMs.

Note that using the "reserve" setting limits the amount of virtual processors that can be shared on a physical machine, and therefore limits the number of concurrent virtual machines you can power on. For example, if you have 20 VMs where each one is configured to have one single virtual processor whose "reserve" value is 100%, and the physical machine has four cores, then you can only power on only four virtual machines. If you try to start a 5th VM, you would get an error message stating that you cannot initialize the virtual machine. Note that you will get this error even if all four running virtual machines are completely idle and are not using any CPU resource (reason mentioned before).

A final interesting note about the virtual machine reserve is that by default it is set to "0%" – which means the reserve is disabled. Furthermore – if you have a configuration like the one where all CPUs are reserved, you can still start extra virtual machines as long as their CPU reserve is set to 0% [13].

Virtual machine limit: This setting is the opposite of "virtual machine reserve" setting. It allows you to specify the maximum amount of processing power that a virtual machine can consume and is relative to the number of processors assigned to a VM.

Relative Weight: It specifies how Hyper-V allocates resources to a VM when more than one VMs are running and compete for resources. This value ranges from 0-10000 and the VM with high value receives more CPU time. By default, all VMs are assigned the same weight value (100).

There is no official document explaining the scheduling algorithms of Hyper-V. Fortunately, Ben Armstrong, virtualization program manager at Microsoft, posted a four part article about CPU scheduling in Hyper-V [13]. From these articles, the scheduling algorithm of Hyper-V can be summarized as follow:

If there is only one VM which has one virtual CPU (vCPU), and the hosting machine has many physical cores, then this vCPU shares the processing resources of all the physical cores. It runs on all the cores in round-robin way. The parent VM is the only VM that stays parked on the first core (core 0) [14].

Thus, Hyper-V uses a weighted round-robin scheduling algorithm. The time a vCPU spends on one physical core before being scheduled to another is not specifically mentioned in any of the public resources. This is something we measured in the testing phase.

B. Xen

Xen is an open source hypervisor developed by Barham et al. [30] at the University of Cambridge in 2003. It is released under the GNU General Public License (GPL), which means that it is open source, free for use and can be modified. It is a bare-metal micro-kernelized hypervisor. Its architectural design is the same as Hyper-V. The parent VM is termed as Domain0 (Dom0 for short) and runs Para-Virtualized (Xen-enabled) Linux OS. This VM helps in creating/configuring guest VMs and has the native device drivers to assist them in performing

real I/O operations [15]. All the other VMs atop Xen are guest VMs and referred as DomUs. DomU PV guests are modified OSs such as Linux, Solaris, FreeBSD, and other UNIX operating systems. DomU FV guests run standard Windows or any other unchanged operating system.

Xen's default scheduler is called Credit [16]. This scheduler acts as a referee between the running VMs. In some ways it's a lot like the Linux scheduler. The Xen team designed it to fairly share the physical CPUs among guest VMs (minimize wasted CPU time). This makes it a "Work-Conserving" (WC) scheduler, in that it tries to ensure that the CPU will always be working, whenever there is work for it to do. As a consequence, if there is more real CPU available than the VMs are demanding, all VMs get all the CPU they want. When there is contention -- that is, when the VMs in aggregate want more CPU than actually exists -- then the scheduler arbitrates fairly between the VMs that want CPU [31].

Credit Scheduler algorithm: Xen is a non-preemptive scheduler based on the WRR (Weighted Round-Robin) scheduling algorithm [17]. It assigns each VM a weight and, optionally, a cap. Weight decides the amount of CPU time it can get and cap fixes the maximum amount of CPU a VM will be able to consume even if the host system has idle CPU cycles. If the cap is 0, then the VM can receive any extra CPU (Work Conserving-mode). A non-zero cap (expressed as a percentage) limits the amount of CPU a VM receives (Non-WC-mode). For example, a cap value of 100 means using only 1 physical CPU, 50 is half a CPU, 400 is 4 CPUs, etc... [18]

Figure 2 is an example of a VM (Domain-0) which has a weight of 256 and cap 0.

```
# xl sched-credit
Cpupool Pool-0: tslice=30ms ratelimit=1000us
Name                                ID Weight  Cap
Domain-0                             0    256    0
```

Fig. 2. Xen VM properties

In the Xen Credit scheduler, every physical core has one Run Queue (RunQ), which holds all the runnable vCPUs (vCPU with a task to run) [16]. The scheduler transforms the weight into a credit allocation for each vCPU, using a separate accounting thread. VCPUs consume credit as they run, and are divided into two categories in the RunQ: UNDER, and OVER. A vCPU is put into the UNDER category if it has remaining credit, and OVER if it runs out of credit. When inserting a vCPU into the queue, it is put after all other vCPUs of equal priority to itself [18]. The scheduler picks vCPUs in the order of UNDER and then OVER. Within each category, it is important to note that vCPUs are scheduled in a round robin fashion. Every 10ms, the scheduler ticks, and then subtracts credits the vCPU owns. When a vCPU consumes all its allocated credits (the value of the credit is negative), the state of its priority changes into OVER, and then the vCPU cannot be scheduled. Every 30ms, the value of credit each vCPU owns is to be accounted again, and all vCPUs will get new value of credit [16]. An example would be a vCPUs waiting in two queues: one for vCPUs with credits and other for those that are over their allocation. Once the first queue is exhausted, the CPU will pull from the second. An IDLE vCPU for each physical core is also created at boot time. It is always runnable and placed at the end of the RunQ. When the IDLE vCPU is scheduled, the physical core becomes IDLE.

Another important setting of the Credit scheduler is called "ratelimit". It is a value in microseconds. It refers to the minimum amount of time a VM is allowed to run without being preempted. The default value is 1000μs (1ms). So if a VM starts running, and a second VM with higher priority wakes up, if the first VM has run for less than 1ms, it is allowed to continue to run until its 1ms is consumed; only after that will the higher-priority VM be allowed to run.

C. VMware ESXi

VMware ESXi (Elastic Sky X) is a bare-metal monolithic hypervisor developed by VMware Inc. It is an operating system-independent hypervisor based on the VMkernel OS, interfacing with agents and approved third-party modules that run atop it [19]. VMkernel is a POSIX-like (Unix style) operating system developed by VMware and provides certain functionality similar to that found in other operating systems, such as process creation and control, signals, file system, and process threads [19]. It is designed specifically to support running multiple virtual machines and provides such core functionality as resource scheduling, I/O stacks, and device drivers [19].

VMware ESXi is the core component of the VMware vSphere software suite which in turn has many software components such as VMware vCenter Server, VMware vSphere Client and vSphere Web Client and many others [20]. All the virtual machines are installed on the physical machine running ESXi hypervisor. To install, manage and access those virtual machines, another part of vSphere suite called vSphere client or vCenter is installed on another physical machine [20]. In earlier versions of VMware vSphere, the hypervisor was available in two forms: VMware ESX and VMware ESXi [32]. The ESXi Server is an advanced, smaller-footprint version of the VMware ESX. Virtualization administrators can configure VMware ESXi through its console or the VMware vSphere Client and check VMware's Hardware Compatibility List for approved, supported hardware on which to install ESXi [21].

Due to its architecture, VMware ESXi uses full-virtualization with Hardware-Assisted approach. The major goals of the VMware CPU scheduler are fairness, throughput, and responsiveness (how promptly a vCPU is scheduled after it becomes ready). ESXi implements a proportional share–based scheduling algorithm, which allocates CPU resources to vCPUs based on their resource specifications. Users can specify the CPU allocation (resource specification) for a VM using three settings: shares, reservations, and limits (Figure 3) [22].

The aim of these three settings is exactly the same as Hyper-V but using different names.

Limit: places a limit on the CPU consumption for a VM. It is specified in MHz (Hyper-V also uses the same word).

Reservation: This is a guaranteed amount of resources reserved for a particular VM. When a VM is powered on, the

reserved amount of resources are given to that VM and are "eliminated" from the pool of open resources that other VMs can use [23]. A VM can (should) not power on if the host it resides on does not have enough resources to meet the reservations of that VM. Its value is expressed also in Mhz. (same naming as Hyper-V)

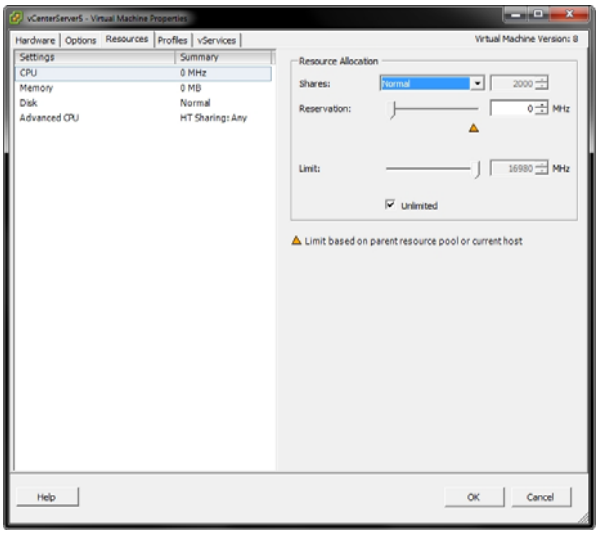

Fig. 3. VM scheduling settings in ESXi hypervisor

Shares: They are simply used to prioritize resources for use by VMs when there is active resource contention. If you give a VM 1000 shares of CPU and another 500, when resource contention arises, the VM with 1000 shares will be granted access to twice as much CPU as the VM with 500 shares. Same idea as relative weight in Hyper-V [23].

Scheduler algorithm: In ESXi, a virtual machine consists of one or more vCPUs on which guest instructions are executed. For example, a 4-vCPU virtual machine has 4 vCPUs. There are other vCPUs associated with the virtual machine that execute management tasks like handling the mouse and keyboard, snapshots, and legacy I/O devices. Therefore, it is theoretically possible that a 1-vCPU virtual machine can consume more than 100% of a processor, although this is unlikely because those management vCPUs are mostly inactive [22].

So if you create a VM on top of ESXi and you assign for it one vCPU, then this VM has in fact many vCPUs, one for executing the guest OS instructions, and other “hidden” vCPUs added automatically by the ESXi scheduler for management purposes. The number of the “hidden” vCPUs is not specified by any documentation.

There are also VMkernel management vCPUs performing management tasks on behalf of the ESXi host. For example, there are a number of vCPUs handling the communication with a management server. A few vCPUs execute periodic tasks for the resource management algorithm. However, the CPU demand of the VMkernel management vCPUs is not significant and most of the CPU resources are available for virtual machines [22].

ESXi CPU scheduler is invoked when the time quantum given to a currently running vCPU expires. Since it is common that a vCPU changes its state to WAIT before the current quantum expires, the size of a time quantum (50 milliseconds by default) does not affect performance much [22]. Time quantum expiration or the state change from RUN to WAIT invokes the CPU scheduler, which then searches for the next ready vCPU to be scheduled. The ready vCPU can be found from a local ready queue or from a remote queue. If none is found, an idle vCPU is scheduled.

When a vCPU wakes up and changes its state from WAIT to READY, likely from an interrupt or a signal, the CPU scheduler is invoked to determine where the newly ready vCPU can be scheduled. It may be put into the ready queue waiting to be scheduled in the next iteration, migrated to a different physical CPU with less contention, or the currently running vCPU may be preempted. The decision depends on many factors like fairness and execution efficiency [22].

III. EXPERIMENTAL CONFIGURATIONS

As mentioned earlier, the three enterprise hypervisors evaluated are: Microsoft (MS) Hyper-V Server 2012R2, Xen 4.2.1 (OpenSuse 12.3 is running in Dom-0 VM) and VMware VSphere ESXi 5.1U1. All these versions were the latest shipping releases at the time of doing this study which started in May 2012.

A. Evaluation hardware platform:

These hypervisors need to run on a physical platform. This platform should meet the hardware requirements for all the chosen hypervisors. Looking at the vendors technical documents [24, 25, 26], the following hardware features are needed:

A 64-bits x86 processor that includes the following:

- At least two cores.
- Hardware-assisted virtualization. This is available in processors that include a virtualization option—specifically processors with Intel Virtualization Technology (Intel VT) or AMD Virtualization (AMD-V) technology.
- Hardware-enforced Data Execution Prevention (DEP) must be available and enabled.
- A Time Stamp Counter (TSC) running at a constant rate, which is the tool used for measuring the time intervals.
- A minimum of 2 GB of physical RAM

To meet these requirements, the most cheap and convenient solution was decided to be the following hardware: 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, 32 KB of L1 instruction cache and 256 KB of L2 cache. L3 cache is 8MB accessible to all cores. 16 GB of physical RAM is used.

B. VM guest OS:

Evaluating the hypervisors is achieved by executing a test suite in a VM running atop each one. This VM is called Under-Test VM (UTVM). As mentioned earlier, Microsoft Hyper-V and Xen support both Full-Virtualization (FV) and Para-Virtualization (PV) VMs, while ESXi supports only FV. For consistency of our evaluation, a guest OS that can be configurable for usage as FV and PV VMs is needed. Being open source and matching the requirements, Linux is chosen. More specifically, a real-time version of Linux called Linux PREEMPT-RT is used. The motive for choosing a real-time kernel is to minimize the latencies that could occur in the guest OS, which in turn produces more accurate measurements. Therefore, Linux PREEMPRT-RT version 3.6.11-rt25 is the UTVM guest OS.

IV. TESTING SETUP

Although the test metrics explained below are mostly used to examine the real-time performance and behaviour of real-time OSs on bare-machines [27, 28], they are useful to be used in other OS test cases. Nowadays, as virtualization together with real-time support is used in an increasing amount of use cases, these tests are a good fit for this paper evaluation.

A. Measuring process

In order to perform our quantitative tests, a tool is needed to measure the time intervals. The cheapest solution is to use an on-processor chip timer running on the constant frequency of the processor clock giving as a value the number of cycles occurred on the processor. Its value is set to zero every time the processor is reset. This timer is called Time Stamp Counter (TSC). It is a 64-bits register present on all x86 processors and has an excellent high-resolution. In order to access the TSC, the programmer has to call the Read Time-Stamp Counter (RDTSC) instruction from assembly language.

B. 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.

1) Clock tick processing duration

Like any time-sharing system, Linux OS allows the simultaneous execution of multiple threads by switching from one thread to another in a very short time frame. The Linux scheduler supports multiple scheduling classes, each using different scheduling algorithms. For instance, there are the two real-time (strict priority) scheduling classes SCHED_FIFO and SCHED_RR, a normal scheduling class (SCHED_OTHER) using dynamic and thus non strict priorities, and finally the SCHED_BATCH class for background threads.

To be able to use timeouts, sleeps, round robin scheduling, time slicing and etc…, some notion of time is needed. On the hardware, there is always a clock responsible for this called the clock timer system. It is programmed by Linux PREEMPT-RT to generate an interrupt each tick. Depending on the kernel configuration used at build time the tick frequency can be selected. We used the 1000 Hz configuration, so that the interrupts occurs each millisecond. This tick period is considered the scheduling quantum.

The aim of this test is to measure the time needed by the OS to handle this clock interrupt. Its results are extremely important as the clock tick interrupt - being on a high level interrupt on the used hardware platform - will bias all other performed measurements. This test helps also in detecting "hidden" latencies that are NOT introduced by the clock tick. In such cases, the "hidden" latency will be different and its event time will not be aligned with the RTOS clock tick frequency.

The test method can be described as follows: a real-time thread with the highest priority is created. This thread does a finite loop of the following tasks: get the time using RDTSC instruction, start a "busy loop" that does some calculations, get 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 only be delayed by the 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. Figure 4 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".

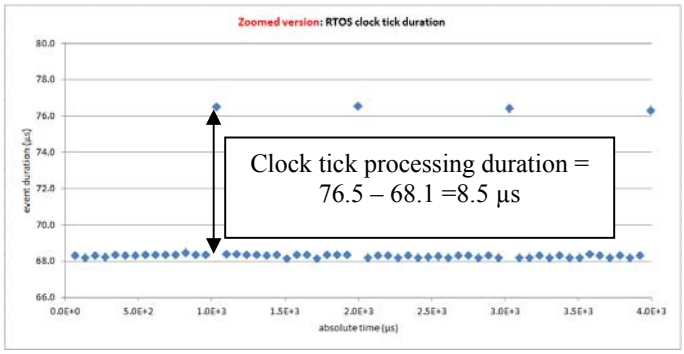

Fig. 4. Clock tick processing duration of the bare-machine

The bottom values (68.1 µs) present the "busy loop" execution durations if no clock tick happens. In case of a clock tick interruption, its execution is delayed until the clock interrupt is handled, which is 76.5 µs (top values). The difference between the two values is the delay spent handling the clock tick interrupt (executing the handler), which is 8.4 µs.

This test detects all the delays that may occur in a system together with its behaviour for a short period. To have a long-time view of the system behaviour, we execute the same test but for a long period. We call the long-duration test as

"Statistical clock tick processing duration". The importance of the figure obtained by "clock tick processing duration" test is to show the exact tracing values and the moments of their occurrence while the figures of "Statistical clock tick processing duration" test (explained below) show their distribution over time and the predictability of the system latencies.

2) Statistical clock tick processing duration

This test is exactly the same as the previous one except that it is done for a longer period. It is executed 5 times, each time for one hour. The motivation for this (5 times) is to take into consideration all the circumstances that may happen in and around the system, like the room temperature, running the test immediately after the machine start-up, run it after one day of keeping the machine on, etc. After 5 tests, the variance was negligible showing the stability of the system. The test with the greatest worst case value was then taken for further processing.

Figure 5 below presents the statistical distribution of the samples obtained during the 1 hour test on the bare-machine. Before looking at the figure results, we first provide an explanation of how we obtain the statistical samples. The measured delay values are counted in binary based bins. This can be done without much overhead as an assembler instruction exists to find the highest bit set in the measured delay. The highest bit set is used as first level bin selection, while the next couple of lower bits are used for the second level bin selection. This makes it possible to statistically collect a huge amount of samples without significant overhead caused by the measurement system itself.

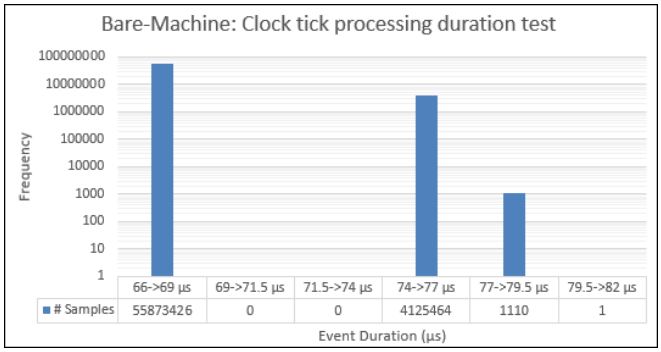

Fig. 5. Statistical clock tick processing duration for a bare-machine

Note that the bin distribution using this method is presented in logarithmic way. Sixty million samples are captured during the 1-hour test. The X-Axis represents the delay values in the binary bins, while the Y-Axis is a logarithmic presentation of the number of obtained samples and their corresponding percentage. For the benchmark provided further on, only the "statistical test" results are used.

After explaining the tests and executing them on the bare-machine, now it is time to run them on the selected hypervisors using several scenarios (use cases).

V. SCENARIOS

Seven scenarios are used in the evaluation of each selected hypervisor. Below is a detailed description of each scenario. It should be reminded that Hyper-V and Xen supports FV and PV VMs. Therefore, in all the scenarios, the evaluation tests are executed in both VMs types, whereas only in FV VM for VMware ESXi hypervisor.

1) Scenario 1: One-To-All (No-Affinity)

The aim of this scenario is to measure the extra overhead added to the VM performance, compared to the bare-machine, due to the insertion of the virtualization layer. Also, we aim to discover the following scheduling behaviour: Does the hypervisor keep the UTVM running permanently on the same CPU or switches it over all the available cores? In this scenario, the UTVM (always with one vCPU) is the only running VM. In case of MS Hyper-V, the UTVM "virtual machine reserve" parameter is set to 100 % (means one physical CPU is dedicated to it). The aim of this parameter is to ensure that no other services (or VMs) share the same processor with UTVM which may influence its performance. A similar parameter (***Reserve***) is set for the UTVM atop VMware ESXi. There is no such parameter for VMs on Xen. Note that in MS Hyper-V and Xen, there is also a parent partition (Dom-0) running but in idle state. The parent partition in Hyper-V runs by default on CPU0 [29]. In Xen, we manually configured it to run permanently on CPU0. The scenario setup is illustrated in Figure 6a.

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

The aim of this scenario is similar to the previous one except that the UTVM vCPU is explicitly assigned to run permanently 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 (latencies) difference (if any) between Affinity and No-Affinity cases. The scenario setup (considering Hyper-V as example) is shown in Figure 6b.

3) Scenario 3: Contention with one CPU-Load VM

This scenario simulates the real life situation where the number of VMs running atop hypervisor can be higher than the available physical cores, which causes resource contention. Its setup has two VMs: UTVM and CPU-Load VM which are both configured to run on the same physical CPU (Figure 6c). 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.

4) Scenario 4: Contention with one Memory-Load VM

Two VMs sharing the same CPU means also sharing the caches. The aim of this scenario is to detect the effect of CPU caches on the UTVM performance. Its setup is exactly the same as scenario 3 except that a Memory-Load VM is used instead of a CPU-Load VM (Figure 6d). This Memory-Load VM is running an infinite loop of memcpy() function which copies 9 MB (a value that is larger than the total cache size) from one object to another. With this memory load, the caches are always flushed.

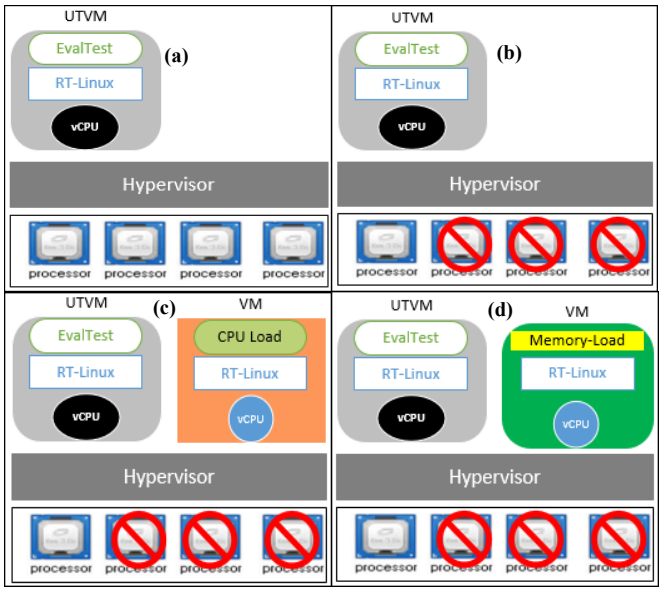

Fig. 6. (a) Scenario 1: One-to-All (b) Scenario 2: One-to-One (affinity) in Hyper-V (c) Scenario 3: Contention with 1 CPU-Load VM atop Hyper-V (d) Scenario 4: Contention with one Memory-Load VM atop Hyper-V

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

In this scenario (Figure 7), 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. In case it is true, then the results of this scenario should be close to the ones of scenario 1. The answer for this is provided in the benchmark table.

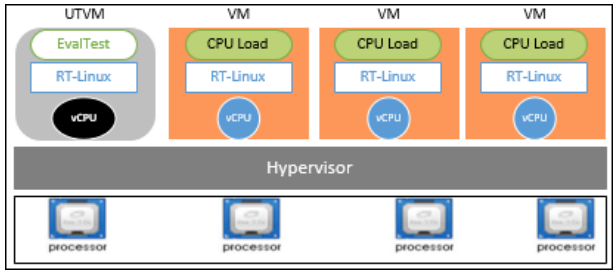

Fig. 7. Scenario 5: All-to-All with three CPU-Load VMs

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

This scenario setup (8) is exactly the same as scenario 5 except using Memory-Load VMs instead of CPU-Load VMs. The aim of this scenario is to clarify whether the type of workload in the VMs has any effect on the performance of the UTVM (Figure 8).

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

This scenario (Figure 9) has two running VMs: the UTVM and a Memory-Load VM. The aim of this scenario and executing it as the last one is justified in the analysis of the benchmark.

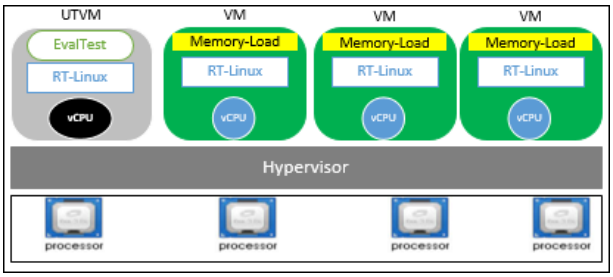

Fig. 8. Scenario 6: All-to-All with three memory-Load VMs

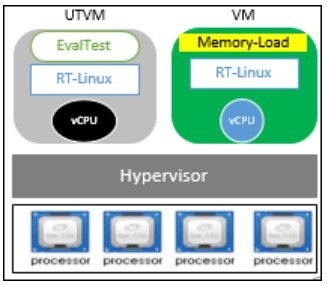

Fig. 9. Scenario 7: Two-to-All with one memory-Load VMs

The tests were executed on the hypervisors using the described scenarios. The results are provided in a benchmark table in the next section together with a detailed analysis.

VI. The Benchmark

This section provides the benchmark table and analysis of the results. Table I shows the tested hypervisors together with the type of used UTVM (Fully-Virtualized or Para-Virtualized) and the experimental scenarios. Microsoft uses the name Emulated for Full-Virtualization, and Enlightened for Para-Virtualization, which are shown in the Hyper-V fields.

A. Analysis of each hypervisor through the scenarios

Hyper-V:

- Our analysis for the scheduling behaviour of Hyper-V especially in scenario 1 showed that its policy is to run a VM on one specific CPU for one second before switching it to the other. This value if obtained by running the short clock test many times. Thus, this value is not obtained from the benchmark table.
- Scenario 2 has a similar setup as scenario 1 except that the VM is explicitly fixed to run permanently on the same CPU. With this configuration, the overheads in scenario 2 are almost double the ones of scenario 1. Therefore, fixing a VM to run all the time on a specific physical CPU is a non-desirable mechanism in Hyper-V.
- The scheduling quantum (the time a VM is allowed to run before being preempted) in case of contention (scenario 3) is nearly doubled (10 ms) for emulated VM compared to enlightened (5 ms). These values are obtained by comparing the values of scenario 2 (the VM is fixed to run on same CPU) and scenario 3.

TABLE I. BENCHMARK TABLE FOR RESULTS ON ALL THE TESTED HYPERVISORS

Maximum overhead by statistical clock tick duration test					
Bare machine value = 10 µs					
Hypervisor / **Scenario**	**Hyper-V Server 2012R2**		**Xen 4.2.1**		**VMware ESXI 5.1**
	Emulated VM	**Enlightened VM**	**Fully-Vitualized VM**	**Para-Virtualized VM**	**Fully-Vitualized VM**
1: One-to-All	4.25 ms	1.62 ms	35 µs	23 µs	59.61 ms
2: One-to-One	7.19 ms	4.08 ms	35 µs	23 µs	185.7 ms
3: Contention with one CPU-Load	18.56 ms	9.16 ms	30.16 ms Customized = 7.44 ms	30.13 ms Customized = 6.74 ms	281.71 ms
4: Contention with one Memory-Load	21.03 ms	11.8 ms	30.18 ms Customized = 8.53 ms	30.15 ms Customized = 7.4 ms	286.5 ms
5: All-to-All with three CPU-Load	5.21 ms	2.55 ms	35 µs	23 µs	51.4 ms
6: All-to-All with three Memory-Load	15.18 ms	4.53 ms	102 µs	62 µs	70.79 ms
7: Two-to-All with one Memory-Load	5.3 ms	2.12 ms	49 µs	32 µs	60.14 ms

- The overheads in scenario 4 are increased by 3 ms due to the effect of the memory load VM on the shared CPU caches. Therefore, the CPU caches have an effect on the system performance (degradation of 3 ms).
- Scenario 5 proved its theory by having very close values compared to scenario 1.
- The results of scenario 6 are around three times greater than the ones of scenario 5 even though the same number of VMs is running. Therefore, this scenario's results confirm our hypothesis about the effect of VM workload type on others performance. The reason for this performance degradation is due to "system memory bus" bottleneck in a Symmetric Multiprocessor System (SMP). i.e. : Comparing the workload of both scenarios (5 and 6), scenario 5 is not causing high overheads because the CPU stress program in the CPU-Load VMs 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 memory bus which in turn will not highly affect the UTVM. In scenario 6, the three Memory-Load VMs are intensively using the system memory 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 three 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, scenario 7 was created where only one Memory-Load VM is sharing the resources with the UTVM. Scenario 7 values clearly show the big performance enhancement if less memory-load VMs are running simultaneously and competing for the system bus.

Xen:

- The values or scenarios 1 and 2 are exactly the same, which means that Xen uses same scheduling policy in both cases (affinity and non-affinity). These values are very close to the bare-machine ones.
- For scenarios 3 and 4, there are 2 values in each field. 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 these two scenarios, we conducted the tests on Xen using the default and minimum scheduler quantum values (30 ms and 1 ms).
- The analysis of the other scenarios is exactly the same as explained for Hyper-V.

The analysis of **VMware ESXi** results is also the same as Hyper-V. The high values in VMs atop VMware ESXi are due to it scheduling policy, which was explained earlier in this paper (section 2, C).

B. General conclusion about each hypervisor

As presented in this paper, MS Hyper-V is tested first. The evaluation results indicated the following findings:

- A PV VM atop Hyper-V performs on average twice better than FV VM. Despite that, PV VM performance is slower than the bare-machine to a big extent.
- The performance of any VM atop Hyper-V is unpredictable with unbounded worst-case latencies (the evaluation test was done 5 times, each with different value).

Xen is tested next, indicating the following findings:

- PV VM atop Xen is on average 1.5 times better than FV VM.
- PV VM performance is very close to the bare-machine.

- VM performance atop Xen is always predictable and with bounded worst case latencies.

VMware ESXi is tested lastly ending up with the following findings:

- VM performance is not comparable to the bare-machine due to the huge difference. Even though, VM performance is predictable with bounded worst case latencies

VII. CONCLUSION

The work done in this paper aims to measure the latencies that happen in a virtualized system, especially when used for hosting real-time applications. To achieve this, a test suite was executed on the top three market hypervisors, VMware ESXi, Microsoft Hyper-V and Xen. These tests are conducted in different scenarios to take into consideration all the parameters and configurations of the hypervisors' schedulers, which are the main sources for influencing the latencies. A benchmark is provided where the results show that the Xen VMs incur the lowest latencies, and its application latencies are comparable to the bare-machine (non-virtualized system) ones. VMs atop VMware incur the highest latencies and is ranked as the last, while Microsoft Hyper-V is the second. This ranking does not intend to eliminate any hypervisor from being qualified for soft real-time applications usage, but gives a clear idea on whether a certain hypervisor can be used for a specific real-time application. Moreover, these values can help users choose the best hypervisor that meets their application requirements. Also, this benchmark provides the scenarios where the best performance of a real-time application can be obtained. Finally, the work shows that latencies in a system are not only software related, but also hardware-related especially in share-memory Symmetric Multiprocessor Systems (SMPs).

REFERENCES

[1] F. Bazargan, C. Y. Yeun, M. J. Zemerly, "State-of-the-Art of Virtualization, its Security Threats and Deployment Models", International Journal for Information Security Research,Vol. 2, No. 3-4, pp. 335-343, 2012

[2] A. Desai, R. Oza, P. Sharma, B. Patel, "Hypervisor: A Survey on Concepts and Taxonomy", International Journal of Innovative Technology and Exploring Engineering, Vol. 2, No. 3, pp. 222-225, 2013

[3] A. J. Younge, R. Henschel, J. Brown, G. Von Laszewski, J. Qiu, G. Fox, "Analysis of Virtualization Technologies for High Performance Computing Environments", IEEE International Conference on Cloud Computing (CLOUD), Washington, USA, pp. 9-16, July 4-9, 2011

[4] VMware, "Understanding Full Virtualization, Paravirtulization, and Hardware Assist", http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf

[5] E. Yuen, "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/

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

[7] J. Hwang, S. Zeng, F. Wu, T. Wood, "A Component-Based Performance Comparison of Four Hypervisors", 13th IFIP/IEEE International Symposium on Integrated Network Management (IM) Technical Session, Ghent, Belgium, pp. 269 – 276, May 27-31, 2013

[8] J. Li, Q. Wang, D. Jayasinghe, J. Park, T. Zhu, C. Pu, "Performance overhead among Three Hypervisors: An experimental study using Hadoop Benchmarks", IEEE International Congress on Big Data, Santa Clara, USA, pp. 9-16, June 27-July 2, 2013

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

[10] Microsoft, "Windows Server 2012 R2", http://www.microsoft.com/en-us/server-cloud/products/windows-server-2012r2/#fbid=NQBnX04C5st

[11] Microsoft Technet Blogs, "Hyper-V: Microkernelized or Monolithic", http://blogs.technet.com/b/chenley/archive/2011/02/23/hyper-v-microkernelized-or-monolithic.aspx.

[12] Virtuatopia, "An Overview of the Hyper-V Architecture", http://www.virtuatopia.com/index.php/An_Overview_of_the_Hyper-V_Architecture.

[13] B. Armstrong, "Hyper-V CPU Scheduling–Part 1 - Ben Armstrong - Site Home - MSDN Blogs", http://blogs.msdn.com/b/virtual_pc_guy/archive/2011/02/14/hyper-v-cpu-scheduling-part-1.aspx.

[14] Microsoft TechNet Articles, "Hyper-V Concepts - vCPU (Virtual Processor)", http://social.technet.microsoft.com/wiki/contents/articles/1234.hyper-v-concepts-vcpu-virtual-processor.aspx?wa=wsignin1.0.

[15] Xen Project Software Overview, http://wiki.xen.org/wiki/Xen_Project_Software_Overview

[16] Linux Foundation, "Credit Scheduler", http://wiki.xen.org/wiki/Credit_Scheduler

[17] S. Yoo, K. H. Kwak, J. H. Jo, C. Yoo, "Toward Under-Millisecond I/O Latency in Xen-ARM", Second Asia-Pacific Workshop on Systems, APSys 2011, Shanghai, China, July 11-12, 2011

[18] X. Xu, P. Sha, J. Wan, J. Yucheng, "Performance Evaluation of the CPU Scheduler in XEN", International Symposium on Information Science and Engineering, pp. 68-72, Shanghai, China, December 20-22, 2008

[19] VMware, "The Architecture of VMware ESXi", http://www.vmware.com/files/pdf/ESXi_architecture.pdf

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

[21] C. Janssen, "What is VMware ESXi Server? - Definition from Techopedia", http://www.techopedia.com/definition/25979/vmware-esxi-server

[22] VMware, "The CPU Scheduler in VMware vSphere 5.1", Performance study-technical report

[23] VMware, "vSphere Resource Management", http://pubs.vmware.com/vsphere51/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-51-resource-management-guide.pdf.

[24] Microsoft, "Hyper-V Overview", http://technet.microsoft.com/en-us/library/hh831531.aspx

[25] Linux Foundation, "Xen Project Beginners Guide", http://wiki.xenproject.org/wiki/Xen_Project_Beginners_Guide.

[26] VMware, "Minimum system requirements for installing ESX/ESXi", http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1003661

[27] L. Perneel, H. Fayyad-Kazan, M. Timmerman, "Android and Real-Time Applications: Take Care!", Journal of Emerging Trends in Computing and Information Sciences, Vol. 4, No. ICCSII, pp. 38-47, 2013

[28] H. Fayyad-Kazan, L. Perneel, M. Timmerman, "Linux PREEMPT-RT vs. commercial RTOSs: how big is the performance gap?", GSTF Journal of Computing, Vol. 3, No. 1, 2013

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

[30] P. Braham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, "Xen and the art of virtualization", SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164-177, 2003

[31] C. Takemura, L. Crawford, in THE BOOK OF XEN, A Practical Guide for the System Administrator, San Francisco, William Pollock, 2010.

[32] WindowsAdmins, "Introduction of vSphere 5 and its components", 2011, http://winadmins.wordpress.com/page/29/