EVALUATING THE PERFORMANCE AND BEHAVIOUR OF RT-XEN

Hasan Fayyad-Kazan[1], Luc Perneel[1], and Martin Timmerman[1, 2]

[1]Electronics and Informatics Department, Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussel, Belgium
[2]Dedicated-Systems Experts Diepenbeemd 5, 1650 Beersel -Belgium

ABSTRACT

Virtualization, together with real-time support emerges to be used in an increasing amount of use cases, varying from embedded systems to enterprise computing. One of the most popular open-source virtualization software's is Xen. Its current implementation does not qualify it to be a candidate for time-critical systems. Researchers and developers extended it and claim the efficient usage of their versions in such systems.

RT-Xen is one of these versions. It is a virtualization platform based on Compositional Scheduling and uses a suite of fixed-priority schedulers.

This paper evaluates the performance and scheduling behaviour of RT-Xen. The test results show that only two proposed schedulers are suitable to be used for soft real-time applications.

1.INTRODUCTION

Virtualization allows the sharing of the underlying physical machine resources between different Virtual Machines (VMs), also called domains, each running its own operating system. The software layer providing the virtualization, inserted between the hardware and VMs, is called the hypervisor or the Virtual Machine Monitor (VMM) [1]. The VMM enables concurrent execution of multiple VMs, isolate them, and schedule them among the available resources.

Virtualization has gained great acceptance and has been widely applied in the enterprise and cloud computing arena. In recent years, it has been deployed in embedded systems domain such as avionics systems and industrial automation where a strong emphasis on real-time performance is required [2]. Also, it has been used for soft real-time applications such as media-based ones which demonstrate inadequate performance due to the impede of virtualization components such as low performance virtualization I/O and increased scheduling latency. [1]
Therefore, the performance overhead introduced by virtualization should be limited.

Current popular virtualization platforms do not support real-time operating systems such as embedded Linux because the platform is not real-time aware [3], which limits the adoption of virtualization in some domains.

Bridging the gap between virtualization and real-time requirements emerges the need of real-time virtualization solutions.

DOI : 10.5121/ijesa.2013.3302

There are many popular virtualization solutions, and one of them is Xen [4]. In its current state, Xen does not support hosting real-time operating systems which require low VM scheduling latency to meet real-time task's deadline. [3]

Recently, efforts were undertaken by researchers [2] to enhance the real-time capabilities of the Xen platform in order to be deployed in a variety of use cases.

The goal of this paper is to evaluate the performance and scheduling behaviour of RT-Xen [5, 6], a real-time version of Xen, which implements a compositional and hierarchical scheduling architecture based on fixed-priority scheduling within Xen.

The rest of the paper is organized as follows: Section 2 is a background description about Xen architecture; section 3 is an overview about RT-Xen; section 4 describes the experimental setup used for our evaluation; section 5 explains the test metrics and scenarios together with the obtained results, and finally a conclusion.

2.XEN: A BRIEF BACKGROUND

A VMM can run either on the hardware directly (called *bare-metal*, or *Type*-1 virtualization), or on top of a host operating system (called *hosted, or Type*-2 virtualization) [2].

Xen is an open-source bare-metal virtualization solution 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) Guest [12] or Fully-Virtualized (FV) Guest [13].

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 [7]. It is lightweight because it can delegate management of guest domains (DomU) to the privileged domain (Dom0) [8]. 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) [7].

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. [7]

Since Xen version 3.0, CREDIT [9] 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, and 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 [6].

3.RT-XEN

RT-Xen project extends Xen, the most widely used open-source VMM, to support real-time systems. It uses hierarchical scheduling theory to bridge the gap between virtualization and real-

time systems by introducing schedulers (servers) in Xen, including a Deferrable Server [10], a Periodic Server [11], and others that are compatible with schedulers in guest operating system. Also, RT-Xen modifies the VCPU parameters that are originally found in Xen by assigning each VCPU with three parameters: priority level, budget and period.

The different schedulers used in RT-Xen differ in the way of managing (consume and replenish) the budget parameter.

Sisu Xi et al. [6] stated: 1) the extensive experimental results demonstrate the feasibility, efficiency, and efficacy of fixed-priority hierarchical real-time scheduling in RT-Xen, 2) the empirical evaluation shows that RT-Xen can provide effective real-time scheduling to guest Linux operating systems at a 1ms quantum, while incurring only moderate overhead for all the server algorithms.

4. EXPERIMENTAL SETUP

The aim of this work is to evaluate the performance and scheduling behaviour of RT-Xen schedulers.

RT-Xen v0.3 [14], being the latest version at the time of evaluation process, is tested here. It is based on Xen v4.0.1. Dom0 is running Xen-modified Linux v2.6.32.13.
To test the real-time capabilities of RT-Xen, a real-time VM running Linux-PREEMPT-RT [15] v3.6.11 is used. This VM is created as being FV with one VCPU. We refer to this VM as Under Test VM (UTVM).

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

5. TESTING PROCEDURES AND RESULTS

5.1 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 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.

5.2 Testing metric

RT-Xen is tested in different scenarios. Before describing the scenarios and the results, we will explain the tests used for evaluation. The first test is called “clock tick processing duration” and executed in the UTVM. We run this test on the bare-machine as a reference before running it in the VM. Below is an explanation of the test.

5.2.1 Clock tick processing duration

This test examines the clock tick processing duration in the kernel. The results are extremely important as the clock interrupt - being on a high level interrupt on the used hardware platform - will disturb 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.

The way we get the clock tick duration in this test is simple: we create a real-time thread with the highest priority. This thread does a finite loop of the following tasks: get time, start a "*busy loop*" that does some calculations, get time again. Having the time before and after the "*busy loop*" provides the time needed to finish the job. In case we run this test on the bare-machine, this "*busy loop*" will only be delayed 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 shows as well when it is scheduled away by the VMM, which in turn impacts latency.

Figure 1 presents the results obtained by 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*".

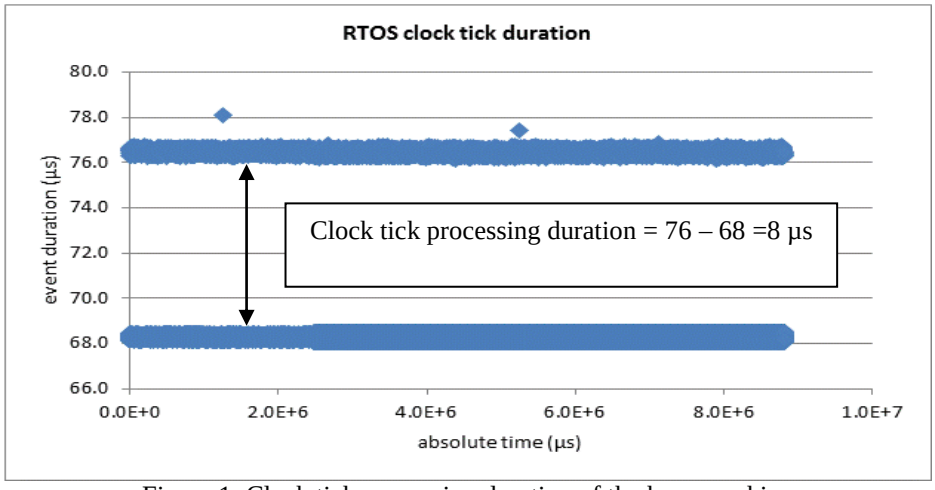

Figure 1: Clock tick processing duration of the bare-machine

The bottom line of figure 1 is the "*busy loop*" execution time if no clock tick happens. As shown, it is 68µs. If a clock tick happens, its execution is delayed until the clock interrupt is handled, which is 76 µs (top line). The difference between the two values is the delay spent handling the tick (executing the handler), which is 8 µs.

At some periods, the "*busy loop*" executes in 78 µs. Therefore, a clock tick delays any task by 8 to 10 µs.

Figure 1 shows 128000 captured samples, in a time frame of 9 seconds. Due to scaling reasons, the samples form a line.

Figure 2 below is a zoomed version of figure 1.

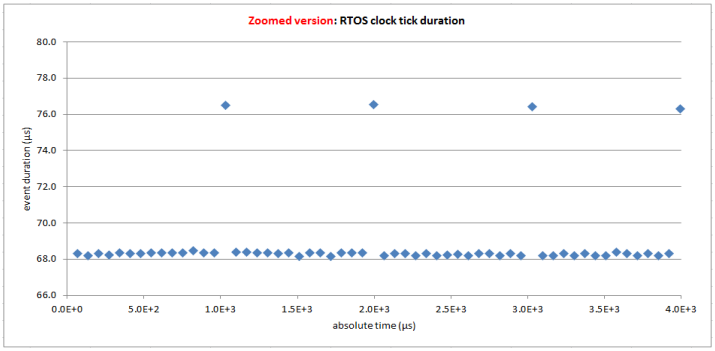

Figure 2: zoomed version of Figure 1

The kernel clock is configured to run at 1000 Hz, Thus, a tick is expected each 1 ms. This is obvious in Figure 2, a zoomed version of figure 1.

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 durations.

This test is done on the bare-machine for 5 hours, and the maximum measurement obtained is 78µs. This means that a worst case overhead of 10 µs occurred in the system during the 5 hours test time.

Now we present the results of executing the same test in the UTVM running on top of RT-Xen.

5.3 Testing RT-Xen

Recall that in *RT*-Xen each VCPU has three parameters: budget, period and priority. When a VM executes, it consumes its budget. A VCPU is eligible to run if and only if it has positive budget. The different servers (schedulers) differ in the way of managing (consuming and replenishing) the budget.

The schedulers (servers) used in RT-Xen v0.3 are:

- *Deferrable server (DS):* If a VCPU has ready tasks, it executes them until either the tasks complete or the budget is exhausted. When the VCPU is idle, its budget is preserved until the start of its next period, when its budget is replenished.[6]
- *Pure Periodic Server (PPS):* In contrast to DS, when a VCPU has no task to run, its budget idles away, as if it had an idle task that consumed its budget.[6]
- *Work Conserving Periodic Server* (WCPS): In the PPS, a VCPU budget is replenished to full capacity every period. The VCPU is eligible for execution only when it has non-empty budget, and its budget is always consumed at the rate of one execution unit per time unit, even if the VCPU is idle. In this server, whenever the currently scheduled VCPU is idle, the VMM's scheduler lets another lower-priority non-idle VCPU run; thus, the system is never left idle if there are unfinished jobs in a lower-priority domain.[5]
- *Slack Stealing Periodic Server* (SSPS): This server utilizes the unused resource budget of an idle VCPU to execute jobs of any other non-idle VCPU, effectively adding extra budget to the non-idle VCPU.[5]

These 4 servers are evaluated in three different scenarios.

5.3.1 Affinity scenario

In this scenario, two VMs (Dom0 and UTVM) are running. Each of them has one VCPU that is pinned to run on different physical core.

The DS scheduler is tested first. Table 1 shows the testing results for this scheduler, followed by analysis of the results.

Task time	Task period	Domain Budget	Domain Period	# Tests performed	Only Tick Overhead	Maximum tick captured	Other Overhead	Maximum overhead captured
7 ms	1 sec	25 ms	50 ms	100	42 %	79 µs	58 %	25.25 ms
7 ms	5 sec	25 ms	50 ms	100	65 %	79 µs	35 %	25.25 ms
7 ms	1 sec	50 ms	50 ms	100	77 %	193 µs	23 %	50.4 ms
7 ms	5 sec	50 ms	50 ms	100	86 %	208 µs	14%	50.4 ms
7 ms	1 sec	75 ms	50 ms	100	100 %	79 µs	0 %	----
12 sec	1 sec	75 ms	50 ms	100	0 %	--	100 %	25.4 ms

Table 1: Testing results of the DS scheduler

The *Task period* column is the delay between running the same task consecutively.

The first row of the table shows that the test time (which has only one task) is 7 ms. We run this test 100 times using a batch file with a *task period* of 1 second. The results show that a maximum measurement of 79 µs is captured in 42% of the tests. This value is due to the clock tick interrupt. A maximum value of 25.25 ms is captured in 58 % of the results which explains that an unknown behaviour other than the clock tick happened in the system.

Normally, as the test time is 7 ms and the VM budget is 25 ms, the test should be completed without any problems or extra overheads.

The reason for having such high measurement could be due to the *task period* value. The test is done again with a *task period* of 5 seconds (2^{nd} row), but the value of 25 ms is still captured.

As the VCPU parameters can have different values, we test the different combinations which are shown in table 1.

Dedicating a VCPU to run alone on a physical core means that it should not be preempted regardless of its parameters. Moreover, this VCPU is assigned a budget greater than the period, which is another way to let it run all the time.

With these configurations, we run a test (always one task) of 12 seconds. The results (last row) show that a value of 25 ms is again captured. Figure 3 shows the behaviour of the system during this test.

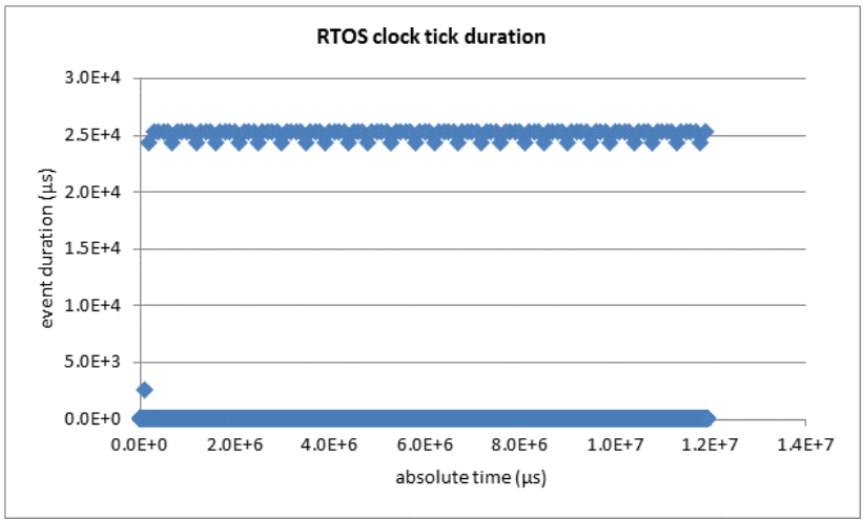

Figure 3: Clock tick processing duration using DS in 12 seconds

As shown, the overhead of 25 ms (the top line) is captured several times during the 12 seconds test. Figure 4 is a zoomed version of the measurement results:

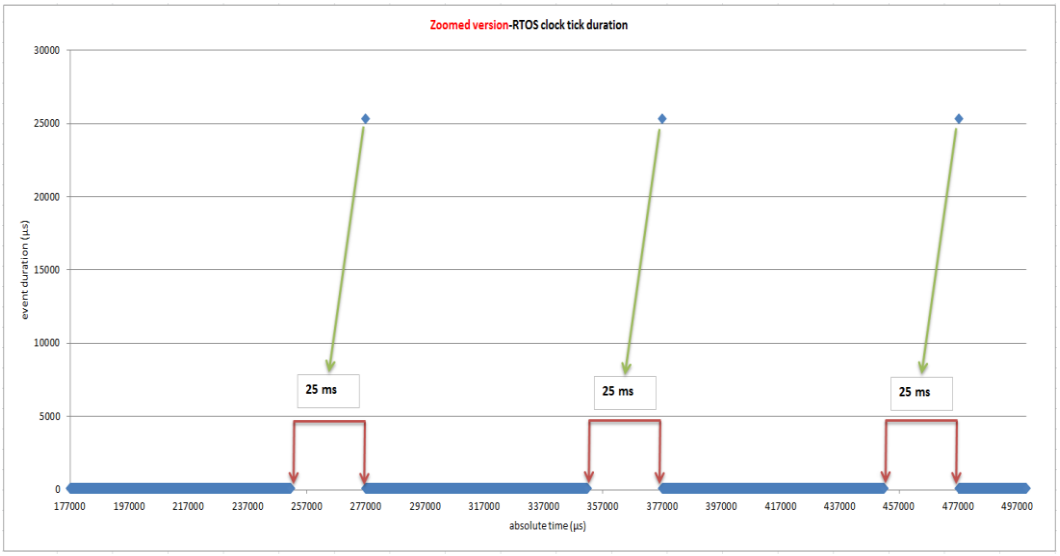

Figure 4: Zoomed version of figure 3

This figure illustrates clearly the behaviour of this scheduler: the UTVM VCPU runs for 75 ms which is its full budget and then gets preempted for 25 ms and so on. Remember that there is no resource contention at all for this VCPU. So scheduling away the only VM requesting the resource is unexpected behaviour. This is a bad scheduler implementation for real-time applications.

As clearly seen in table 1, the *task period* is not the cause for bad behaviour. Thus, the other servers are tested only with the one second *task period.*

Below are the results of the other three supported servers:

Task time	Task period	Domain Budget	Domain Period	# Tests performed	Only Tick Overhead	Maximum tick captured	Other Overhead	Maximum overhead Captured
7 ms	1 sec	25 ms	50 ms	100	73 %	79 µs	27 %	25.3 ms
7 ms	1 sec	50 ms	50 ms	100	92 %	78 µs	8 %	1.12 ms
7 ms	1 sec	75 ms	50 ms	100	100 %	78 µs	0 %	---
6 sec	1 sec	75 ms	50 ms	100	100 %	90 µs	0 %	---
687 sec		75 ms	50 ms	1	100 %	82 µs	0 %	---
3489 sec (50 million samples)		75 ms	50 ms	1	100 %	106 µs	0 %	---

Table 2: Testing results of the Pure Periodic Server (PPS)

Task time	Task period	Domain Budget	Domain Period	# Tests performed	Only Tick Overhead	Maximum tick captured	Other Overhead	Maximum overhead Captured
7 ms	1 sec	25 ms	50 ms	100	69 %	78 µs	31 %	25.25 ms
7 ms	1 sec	50 ms	50 ms	100	95 %	80 µs	5 %	1.14 ms
7 ms	1 sec	75 ms	50 ms	100	100 %	79 µs	0 %	---
6 sec	1 sec	75 ms	50 ms	100	100 %	80 µs	0 %	---
687 sec		75 ms	50 ms	1	100 %	79 µs	0 %	---
3489 sec (50 million samples)		75 ms	50 ms	1	100 %	109 µs	0 %	---

Table 3: Testing results of the Work Conserving Periodic Server WCPS

Task time	Task period	Domain Budget	Domain Period	# Tests performed	Only Tick Overhead	Maximum tick captured	Other Overhead	Maximum overhead Captured
7 ms	1 sec	25 ms	50 ms	100	63%	95 µs	37 %	25.7 ms
7 ms	1 sec	50 ms	50 ms	100	95 %	95 µs	5 %	1.14 ms
7 ms	1 sec	75 ms	50 ms	100	100 %	94 µs	0 %	---
6 sec	1 sec	75 ms	50 ms	100	100 %	105 µs	0 %	---
687 sec		75 ms	50 ms	1	100 %	106 µs	0 %	---
3489 sec (50 million samples)		75 ms	50 ms	1	100 %	108 µs	0 %	---

Table 4: Testing results of the Slack Stealing Periodic Server SSPS

A summary of this scenario shows that if the VCPU budget is lower than or equal its period, none of the servers behave correctly.

If a VCPU budget is greater than its period, all servers behave correctly *except DS.*

5.3.2 Contention Scenario with 1 RT VMs

Recall that in the previous scenario, all servers *except* the Deferrable Server behaved correctly whenever the VCPU *budget* is greater than its *period*. Therefore, DS is not tested here.

The aim of this scenario is to detect the quantity of overhead that affects the UTVM while sharing the same resource with another RT VM.

Both, the UTVM and another RT VM (RT-Domain 1) share the same core, while Dom0 is running alone on another core.

Each of the 2 VMs has a budget greater than the period. The results of the 3 servers are shown in the tables below:

Task time	# Tests done	UTVM					RT-Domain 1			
		Budget	Period	Priority level	Maximum overhead	Executed in:	Budget	Period	Level	State
20.6 sec	10	500 ms	400ms	10	1.15 ms (100%)	42.65 sec	500 ms	400 ms	10	IDLE
20.6 sec	10	75 ms	50 ms	10	1.15 ms (100 %)	42.65 sec	75 ms	50 ms	10	IDLE
20.6 sec	10	75 ms	50 ms	1	81 µs (100%)	20.6 sec	75 ms	50 ms	10	IDLE
20.6 sec	10	75 ms	50 ms	10	1.13 ms (100%)	41.5 sec	75 ms	75 ms	10	Running
20.6 sec	10	75 ms	50 ms	1	82 µs (100 %)	20.6 sec	300 ms	200 ms	10	Running

Table 5: Testing results for the PPS in scenario 2

The first row in table 5 shows that a task of 20.6 seconds executes on the UTVM whose priority level is 10. RT-Domain1 is also of same priority level and in IDLE state. This test is done 10 times, and a maximum measurement of 1.15 ms is captured in the 10 times (100%). The test is executed in 42.65 seconds instead of 20.6 seconds, which is an increase of 107 %.

The values in the 2nd row are the same as the first except for the budget and period. These values are changed to test if they cause any effect on the results. Clearly, the answer is NO.

Prioritizing the VMs by assigning a high priority (low value means higher priority) to the UTVM resulted in a maximum measurement of 81 µs, and the task finished in its time.

WCPS scheduler is tested now.

Task time	# Tests performed	UTVM					RT-Domain 1			
		Budget	Period	level	Maximum overhead	Executed in:	Budget	Period	Level	State
20.6 sec	10	75 ms	50 ms	10	110 µs 100%	21.4 sec	75 ms	50 ms	10	IDLE
20.6 sec	10	75 ms	50 ms	1	81 µs 100%	20.6 sec	75 ms	50 ms	10	IDLE
20.6 sec	10	75 ms	50 ms	10	1.14 ms 100 %	42.74 sec	75 ms	50 ms	10	Running
20.6 sec	10	75 ms	50 ms	1	82 µs 100 %	20.6 sec	75 ms	50 ms	10	Running

Table 6: Test results of WCPS

In table 6, if RT-Domain1 is idling, WCPS scheduler behaves better than PPS server.

If RT-Domain1, with the same priority level as UTVM is running, then same maximum measurement is obtained in both schedulers. Figure 4 illustrates the behaviour of WCPS in this case.

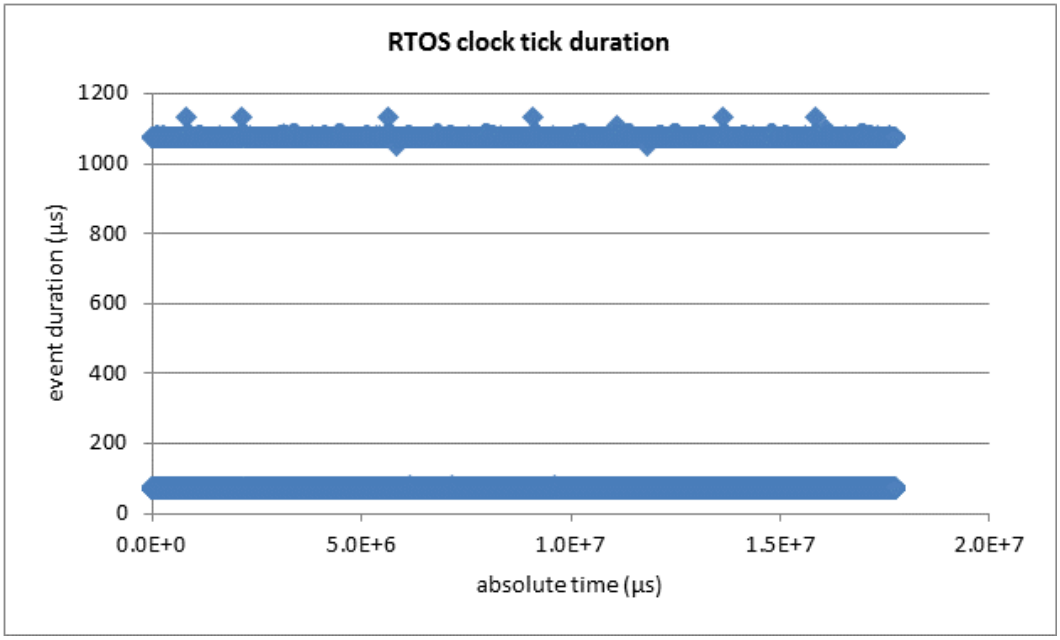

Figure 5: Clock tick test of WCPS showing the overhead of 1 ms

It is clearly visible that the measurement of 1.14 ms is captured a lot. Figure 5 is a zoomed version of figure 4 to have a clear view of the behaviour.

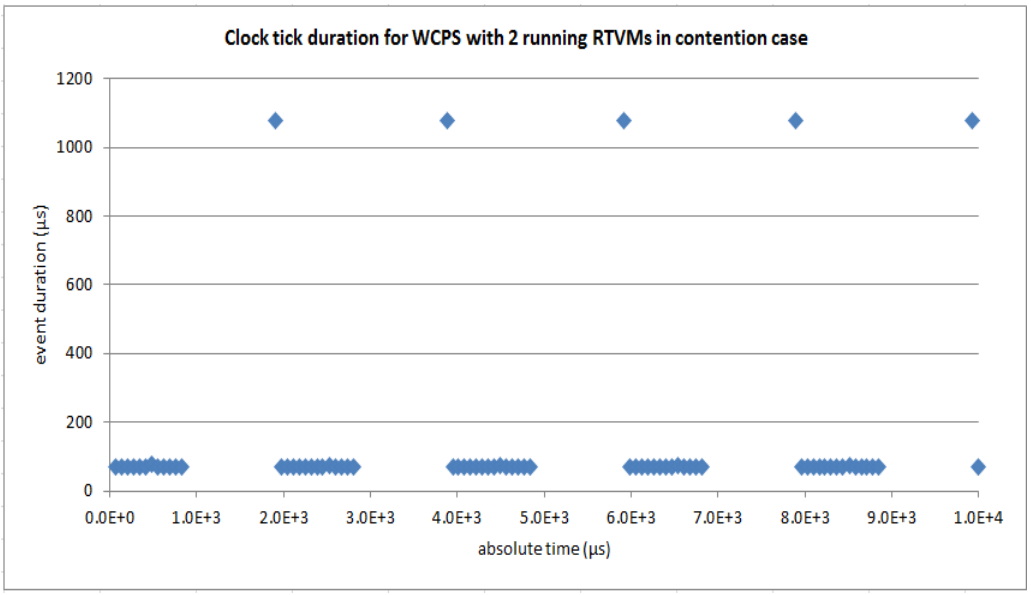

Figure 6: Zoomed version of Figure 4

Figure 5 shows that the VCPU in preempted each 2ms for a duration of 1 ms.
Below are the test results of the SSPS scheduler.

Task time	# Tests performed	UTVM					RT-Domain 1			
		Budget	Period	level	Maximum overhead	Executed in:	Budget	Period	Level	State
20.6 sec	10	75 ms	50 ms	10	110 µs 100 %	21.7 s	75 ms	50 ms	10	IDLE
20.6 sec	10	75 ms	50 ms	1	83 µs 100%		75 ms	50 ms	10	IDLE
20.6 sec	10	75 ms	50 ms	10	1.14 ms 100 %	42 sec	75 ms	50 ms	10	Running
20.6 sec	10	75 ms	50 ms	1	82 µs 100 %		75 ms	50 ms	10	Running

Table 7: Test results of SSPS

The behaviour and performance of SSPS is similar to WCPS.

A summary of the results show that if two VMs of same priority level are running on a shared resource, then the scheduling overhead of the three schedulers is almost the same. If one VM is running while the other is idle, the PPS scheduling overhead is higher than WCPS and SSPS.
In all the servers, a VM with higher priority level is not affected by any scheduling overhead.
Therefore, WCPS and SSPS are the best to be used in case of contention scenario.

In the next scenario, only these 2 servers are tested, as it is again a contention scenario with more VMs.

5.3.3 Contention scenario with 2 RT VMs

This scenario is the same as the previous, except that three RT VMs runs on the same core (the UTVM and two others). Any of these VMs can be running or idling. As stated before, only WCPS and SPSS are tested as they perform better than PPS.

Below is the analysis of the tests on each of the two servers:

Task time	# Tests done	UTVM					RT-Domain 1 RT-Domain 2			
		Budget	Period	level	Maximum overhead	Executed in	Budget	Period	Level	State
20.6 sec	50	75 ms	50 ms	10	150 µs	21.5 sec	75 ms	50 ms	10	Both IDLE
20.6 sec	50	75 ms	50 ms	10	1.16 ms	26.5 sec	75 ms	50 ms	10	1 IDLE 1 Running
20.6 sec	50	75 ms	50 ms	10	2.14 ms	62.7 sec	75 ms	50 ms	10	Both Running

Table 8: Test results for WCPS

In the previous scenario, for the same scheduler, while only RT-Domain1 is running, a maximum value of 1.14 ms is captured in UTVM. Having RT-Domain1 and RT-Domain2 running, a maximum value of 2.14 ms is captured, which is logical.

The same behaviour happens in idling state. Recall that in the affinity scenario, the maximum captured measurement is 80 µs. In the scenario of contention with RT-Domain1, the maximum

captured measurement is 120 µs. Having RT-Domain1 and RT-Domain2 sharing the resource with UTVM, a maximum value of 150 µs is captured (row 1) which means that an additional VM causes an extra overhead of 30 µs.

The SSPS scheduler is tested now.

Task time	# Tests done	UTVM					RT-Domain 1 RT-Domain 2			
		Budget	Period	level	Maximum overhead	Executed in	Budget	Period	Level	State
20.6 sec	50	75 ms	50 ms	10	160 µs	21.5 sec	75 ms	50 ms	10	Both IDLE
20.6 sec	50	75 ms	50 ms	10	1.2 ms	42.6 sec	75 ms	50 ms	10	1 IDLE 1 Running
20.6 sec	50	75 ms	50 ms	10	2.14 ms	62.7 sec	75 ms	50 ms	10	2 Running

Table 9: Test results of SSPS

SSPS scheduler results show similar behaviour as WCPS scheduler.

5.3.4 Response time in case of interrupts

The aim of this test is to detect the time required by an idling high priority VM to respond for an interrupt. It is applied for Scenario 2. In this test, the RT-Domain1 is running, while UTVM is idling and having a higher priority. The question to be answered after executing this test is: Will the UTVM respond immediately to the interrupt or it waits until the budget of RT-Domain1 is exhausted or has no task to run.

To perform this test, an external PCI device which generates interrupts using an internal timer is inserted in the machine. PCI Pass-through is used to assign this device to the UTVM in order for the interrupts to be captured and handled by this VM. The duration between successive generated interrupts is 0.5 ms.

This test is done using the three scheduler of Scenario 2; PPS, WCPS and SSPS.

Figure 6 presents the results of PPS scheduler.

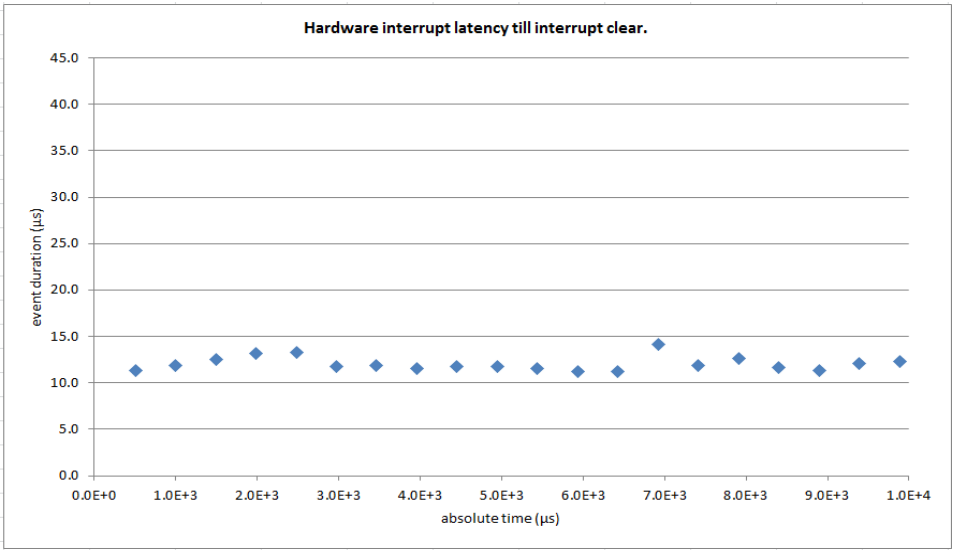

Figure 7: Time required handling an interrupt in the VM

As shown in figure 6, the time required to handle the interrupt in UTVM is 14 µs. This time duration includes the scheduling of UTVM by the PPS scheduler and then executing the interrupt handler inside it. The figure above shows the results of a short test period. We did this test for a large period (8 seconds) for having more samples. The results of all the tested schedulers are shown in table 10.

Scheduler	Interrupt handling duration
PPS	41.5 µs
WCPS	21.2 µs
SSPS	21.2 µs

Table 10: Clock handling duration for the three tested schedulers

The table results show that for the three tested schedulers, the UTVM responded immediately to its interrupt even the other VM still has task to do. This is a good behaviour for real time systems.

6. CONCLUSION

Xen, the most popular open-source virtualization software, does not support real-time systems. RT-Xen, an extended version of Xen for supporting such systems, is tested here.

Xen default scheduler, called credit scheduler, is designed to ensure proportionate sharing of CPUs. In this scheduler, each Virtual CPU (VCPU) is assigned a parameter called the weight, and CPU resources (or *credits*) are distributed to the VCPUs of the VM in proportion to their weight.

In turn, RT-Xen instantiates a suite of fixed-priority schedulers or servers including Deferrable Server (DS) , Pure Periodic Server (PPS), Work Conserving Periodic Server (WCPS) and Slack Stealing Periodic Server (SSPS).

In RT-Xen, each VCPU has 3 parameters, level (priority), budget and period. When a VM executes, it consumes its budget. A VCPU is eligible to run if and only if it has positive budget. The different servers (schedulers) differ in the way of managing (consuming and replenishing) the budget.

RT-Xen schedulers are tested in three scenarios.
The first scenario is called *Affinity scenario*, where only one VM with one VCPU runs on a physical core. The test results show that none of the servers behave correctly if the VCPU budget is lower than or equal to its period. With the VCPU budget greater than the period, all servers *except* the Deferrable server behave correctly. For that, DS scheduler is excluded in the next scenarios.

The 2nd and 3rd scenarios are called *Contention scenarios*. In these scenarios, the Under Test VM (UTVM) shares the resource (core) with another RT VMs. The test results show that while the RT VMs are running simultaneously with UTVM, the scheduling overhead imposed by all the schedulers is almost the same.

If one or more RT VMs are in idle state, the UTVM overhead imposed by using PPS scheduler is much higher than WCPS and SSPS schedulers.

Thus, WCPS and SSPS servers prove to be the best for usage in all the scenarios.

A final test evaluated the response of UTVM, having a high priority, in case of interrupt occurrence in contention scenario. The aim of this test is to see if the UTVM responds immediately to an I/O interrupt or waits until the time slice of contenting VMs is exhausted in order to handle the interrupt. Schedulers of contention scenarios passed this test and showed good behaviour.

This evaluation shows that RT-Xen is a good virtualization solution for soft real-time applications if some configuration precautions (depending on the use case) are considered. RT-Xen founders are still enhancing it, which represents it as promising software in real-time virtualization arena.

REFERENCES

[1] Min Lee, A. S. Krishnakumar, P. Krishnan, Navjot Singh, Shalini Yajnik "Supporting Soft Real-Time Tasks in the Xen Hypervisor" Proceeding VEE '10 Proceedings of the 6th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, 2010.

[2] Zonghua Gu, Qingling Zhao "A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization" Journal of Software Engineering and Applications, 2012

[3] Peijie Y., Mingyuan X., Qian L., Min Z., Shang G., Zhengwei Q, Kai C., Haibing G. Real-time Enhancement for Xen hypervisor, IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC), 2010

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

[5] J. Lee, S. S. Xi, S. J. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Y. Lu and O. Sokolsky, "Realizing Compositional Scheduling through Virtualization," *Technical Report*, University of Pennsylvania, Philadelphia, 2011.

[6] S. S. Xi, J. Wilson, C. Y. Lu and C. Gill, "RT-Xen: Towards Real-Time Hypervisor Scheduling in Xen," *Proceedings of the* 2011 *International Conference on Embedded Software*, Taipei, 9-14 October 2011, pp. 39-48.

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

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

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

[10] J. Strosnider, J. Lehoczky, and L. Sha. The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real Time Environments. IEEE Transactions on Computers, 1995.

[11] L. Sha, J. Lehoczky, and R. Rajkumar. Solutions for Some Practical Problems in Prioritized Preemptive Scheduling. In RTSS, 1986.

[12] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles* (2003), ACM, p. 177.

[13] Devine, S., Bugnion, E., and Rosenblum, M. Virtualization system including a virtual machine monitor for a computer with a segmented architecture, May 28 2002. US Patent 6,397,242.

[14] RT-Xen, Downloads for RT-Xen, https://code.google.com/p/rt-xen/downloads/list

[15] Linux Foundation, Real-Time Linux Wiki, https://rt.wiki.kernel.org/index.php/Main_Page