Impact of Machine Virtualization on Timing Precision for Performance-critical Tasks

Kirill Karpov¹, Irina Fedotova¹ and Eduard Siemens¹

¹Anhalt University of Applied Sciences - Faculty of Electrical, Mechanical and Industrial Engineering, Bernburger Str. 57, 06366, Köthen, Germany

kirill.karpov@hs-anhalt.de

Abstract. In this paper we present a measurement study to characterize the impact of hardware virtualization on basic software timing, as well as on precise sleep operations of an operating system. We investigated how timer hardware is shared among heavily CPU-, I/O- and Network-bound tasks on a virtual machine as well as on the host machine. VMware ESXi and QEMU/KVM have been chosen as commonly used examples of hypervisor- and host-based models. Based on statistical parameters of retrieved distributions, our results provide a very good estimation of timing behavior. It is essential for real-time and performance-critical applications such as image processing or real-time control.

1. Introduction

Modern computer systems are often virtualized to sufficiently increase the hardware utilization, to raise the flexibility of system configuration and to reduce maintenance costs. However these advantages come with several challenges due to the virtualization concept, which can dramatically impact the support of real-time and high precision applications. While the host operating system controls the access to the physical resources directly, the guest OS has to traverse through the host operating system layer before it can access the resources. This additional software layer impacts performance and may be critical for applications which require precise and accurate timing functionality such as video streaming, video games or medical surgery equipment control.

Keeping time precision of control algorithms and meeting soft real-time requirements under virtual environments is a complex problem that requires efficient access to the physical platform [1]. Typically the peripheral hardware of computing system is not directly accessible by the user-level application software layers or even the host side without any form of virtualization. Thus, timing operations provided by the OS are too inefficient for real-time tasks [2]. While on Linux direct access to the TSC counter can improve context switching on Linux by avoiding expensive system calls for time acquisition, precise delay measurements are a more challenging task. In fact, timer access through the RDTSC machine instruction available for x86-64 architecture [3, 4] can cancel out the speed benefits by invoking a call up to 2-3 times. However, a sleep procedure, whose realization in the standard GNU C library has an insufficient precision due to the long wake-up time, can even more aggravated when sharing resources between host and virtual machines (VM).

While applying different techniques for timekeeping on Linux machines, we intend to demonstrate problems that can arise for time-critical applications running in a virtualized environment. Two basic [5] virtualization classes have been studied: the hypervisor-based model (with VMware ESXi) and the
hosted-based model (QEMU/KVM). The hypervisor-based model, also known as virtual machine
monitor (VMM), is essentially a tiny operating system, which allocates hardware resources among VMs.
In contrast, hosted-based implementations use already present operating system resources and together
with a virtualization module act as a VMM. In this case, the host OS integrates a VMM layer that is
responsible for providing the VMs with their virtual platform interface and managing all context
switching and scheduling. Such segregation will obviously add additional overhead in reaction to the
timer interrupts and can increase the cost for time measurement. Therefore, considering the difference
between communication with the underlying physical hardware and resource management, the
quantitative and qualitative analysis of timing behavior on both models is essential for the planning of
real-time applications on virtualized systems. Moreover, the comparison should include not only the
proof how fast the hardware access from guest operating system level can be, but also how precisely a
system wakes up from a sleep operation in a virtual environment.

The remainder of this paper is structured as follows: Section 2 reviews related work. Section 3 is
devoted to the experimental setup, describes hardware and software equipment, load types, measurement
and representation methods for the current research. The results of experiments with time acquisition
and sleep function measurements is presented in Section 4 and 5 respectively. Finally Section 6 discusses
the results, followed by the future work and the conclusion in Section 7.

2. Related Work
A detailed performance analysis of Xen Linux was made by P. Barham [6]. The authors made
performance measurements in the hosting scenario on the Xen hypervisor-based VMM with different
loads to simulate real life servers and compared it with different virtualization software.

A deep analysis of influence of visualization on network performance on Amazon EC2 service was
performed in [7]. The authors have considered a wide number of network parameters such as Bandwidth,
RTT, delay, packet loss rate, throughput and provided early results towards understanding the
characteristic of Xen-based virtualized data centers.

S. G. Langer and T. French [8] have conducted research of influence of virtualization on I/O
operations. It has been performed on Red Hat, Xen, ESXi, kvm_redhat64 as hypervisors, VMWare
player and VirtualBox as hosted-based VMMs.

Moreover, a few studies have investigated the impact of machine virtualization on time related
functionality. For instance, authors in [9] have done a comprehensive analysis of different time
acquisition methods. They simulated idle, CPU and I/O-loaded systems on a host machine and partly
compared the results on five different virtual platforms: VMware Player, VirtualBox, QEMU, Xen, and
KVM. However, they performed this without observation of sleep precision.

The innovation novelty of our work is the full investigation of virtualized timer hardware, which
includes not only measurements for time acquisition, but also the reaction of timers on waiting
operations. Moreover in this work, we intended to create conditions as close as possible to real world
applications running on virtual systems, simulating different kind of loads on both host and guest sides.

3. Experimental setup

3.1. System configuration
All tests are performed on the same hardware platform, a simple hardware server-like set with a main
requirement to CPU to include Intel-VT/AMD-V Virtualization Technology (VT) support [10]. For the
test Intel(R) Core(TM) i7-860 @ 2.80GHz (Lynnfield) with 4 physical cores along with 8 GB of RAM
and Broadcom BCM5751 Netxtreme Gigabit NIC have been used. All measurements on both host and
guest operating systems have been performed on Ubuntu 14.04.4 LTS 4.2.0-34-generic kernel with
standard scheduling configurations. Measurements have been performed on the system with invariant
TSC counter with a constant rate. Hyper-threading mode has been disabled to derive a serialized
execution. Virtual machines have been configured with enabled VT technology to replicate physical
hardware, and were provided with two CPUs and 1 GB of RAM each.
3.2. Load types
In the context of multiprogrammed computing system, it’s possible to distinguish several main types of tasks [11, 12]. According to this classification, we distinguish the following load scenarios and create corresponding testbeds for the experiment:

*Idle testbed* – in this load type operating system works in normal mode, without any additional working applications. An idle system represents the best-case scenario.

*CPU testbed* – this type creates very large CPU load using up to 100 % of available CPU resources. The stress utility creates 100 threads with square root calculations. This type of load is often created by using any video/audio conversion/compresion algorithms or applications for graphic processing.

*IO/Network testbed* – this load type simulates typical VMs load, which can exist on hosting in a large data center. It creates NFS server on VM and starts sending and receiving data calling standard read/write calls. This workload results in a large number of context switches and disk interrupts. As an example, this scenario can be used for solving such problems like distributed processing of big data.

*Network testbed* – in this mode a VM instance runs a Reliable Multi Destination Transport Protocol (RMDT) [13], which runs in bi-directional transmission mode. For the current testbed RMDT has been configured to send repeatedly the same chunks of data from a buffer to avoid stalling on disk IO. The protocol fills all available bandwidth, converging to the limit of network internet card and allows to create an intensive network load in long fat pipes manner.

3.3. Measurement Methodology
First, the experiment evaluates the cost of setting a timer: the time required to obtain a time value from the timing hardware. We use the term *timer cost* as a measuring parameter. Second, the measurements of sleeping operations have been performed and the difference between the targeted wake-up time and actual times of wake-ups (referred as a *miss time*) have been calculated. For both cases of measuring both the *timer cost* and *miss time*, the usage of system call is contrasted to gathering counter cycles from timer hardware. As an interface to available time counters the HighPerTimer library has been applied [14]. It is used to obtain access to the TSC timer on the current processor from user-space and provides a mechanism for precise sleep operations. The sleep function from HighPerTimer library (*HPTSleep*) allows to significantly decrease the overhead of miss time without starving the CPU.

In present research, the median has been chosen as the main statistical significative. Since the data follows a non-parametrical distribution, the median value carries more information about distribution than the mean value. In the next sections we use two types of plots: plots with raw data with bold lines which are the medians and plots with the representation of Complementary Cumulative Distribution Function (CCDF) with dashed lines which are the means. CCDF is a distribution function that shows how often the random variable is above a particular level. Additionally, the mean values are illustrated within CCDF plots, which allow to estimate how mean values are deviated from medians.

4. Experimental Results For Time Measurements
Time measurements are performed with TCS timers and clock_gettime() system call in three scenarios: Host OS, where operating system is placed on bare hardware, VMware ESXi for hypervisor-based virtualization and QEMU for virtual monitor based virtualization. When acquiring TSC timer data through the RDTSC machine instruction, we will henceforth refer to it as the TSC timer method. Similarly, time acquisitions using the clock_gettime() system call is referred to as system call.

4.1. Idle testbed
Initially all measurements have been performed in idle state in order to compare the results with heavy loads on the next stages. As shown in figure [1], median values on all three environments are almost the same with TSC timers as well as with system calls. In all cases TSC timers show better performance than system call. As expected, the host OS has the lowest time overhead (figure [1] (a)), about 17.2 nsec when using the TCS timer and 27 nsec for the system call. VMware ESXi (figure [1] (c)) shows lower precision than the host OS as well as a deviation and at the same the time mean value is lower than with
QEMU, 20.1 nsec for TSC timers and 34 nsec for the system call. TSC timers on QEMU virtualized OS (figure [1] (b)) have a median value near to the idle value, 21.5 nsec, however the system call has a significantly larger overhead, 66 nsec. Figure [2] shows that data from all sources have quite similar distributions. Comparing results of forwarding of the RDTSC instruction on VMware ESXi and QEMU, VMware shows less overhead on the translation. Considering the CCDFs of giving results that presented in figure [2], it can be established that TSC timer without load can guarantee overhead up to 20 ns with probability 0.999. Also blue and red lines on the plot which correspond to QEMU are grouped together, they have the longest tails in worst case scenario.

4.2. CPU-bound Load

CPU-bound background workload can be expected to result in worsened latencies due to limited hardware resources such as memory bandwidth and shared caches [11]. The results of the experiment with CPU-bound load are shown in figures [3] and [4]. They show that at there is no big effect on median values. However it causes long tails in the CCDF plot, which implies significant influence on the mean of the distributions. The graphs are shifted to the right, to the region of 1 µs. In this experiment, we can establish that probability of exceeding out 20 nsec is lower than $10^{-3}$ with TSC timer. In other words, we can guarantee that time overhead will be not greater than 20 nsec in 99.99% of all cases.

4.3. IO/Network-bound Load
Further, challenges of interrupt handling caused by IO-bound tasks and network load are illustrated in figures 5 and 6. Gathering values from TSC shows almost native overhead on all platforms. Although VMware ESXi has median values (figure 5 (c)) of data gathered with TSC timer higher than Host OS and QEMU, it has significantly lower spread than QEMU as well as Host OS. This also can be seen in figure 6, the tails of distributions from VMware ESXi datasets indicate lower time cost value than in other cases. Nevertheless, the virtualized TSC timer on QEMU shows an unexpected advantage: its median overhead value even lower than median value obtained on the same non-virtualized hardware where QEMU placed on. This can be seen on CCDF plot (figure 6 blue line), 90% of outcomes are concentrated below 17 nsec and past this value the distribution line falls steeply and then follows the behavior of other TSC distributions. Also with this plot the region of reliable operation can be determined. Under such load TSC can guarantee that 99.999% of outcomes will not exceed 21 nsec.

![Figure 5: Measurements of timer cost for the IO/Network testbed on: (a) Host (b) QEMU (c) VMware ESXi](image)

![Figure 6: CCDF representation of measurements of timer cost for the IO/Network testbed](image)

4.4. Network-bound Load

The results of the given experiment are presented in figures 7 and 8. In this case, like in previous experiments, the TSC timers show lower time overhead than the system call. However, as seen in figure 7, similarities in measurement results are shared mostly between HPTimer and the system call, rather than between different virtualization environments. Figure 8 shows that these similarities mostly concern the tails of the distributions. Long and heavy tails of distributions that have been obtained with system calls and the TSC counter are grouped together according to the platform. Although distributions with VMware ESXi have the longest tails, TSC timer shows the lowest median value and system call has median value lower than system call on QEMU. Also, based on CCDF plots we can estimate that 99.99 % of all outcomes, obtained with TSC timer during the experiment have values less than 22 nsec.
5. Experimental Results For Sleep Measurements

Sleep measurements are also performed by comparing the sleep function of the HighPerTimer library and the `usleep()` call of the standard C library in three scenarios: Host OS, VMware ESXi, and QEMU. Experiments with the sleep operation from HighPerTimer library are called `HPTSleep` in this paper. Experiments with `usleep()` form standard C library are called `uSleep`.

5.1. Idle testbed

As in the previous section, measurements have to be performed in idle state in order to establish a baseline when comparing load scenarios. As shown in figure 9, median values during `HPTSleep` function on all three platforms are in the same range—between 80 and 116 nsec. The invocation of `uSleep` system call shows miss times that are almost three orders of magnitude greater than `HPTSleep` function as expected: 60 µsec compared to 80 nsec on Host OS, 97 µsec compared with 116 nsec on QEMU, and 63.1 µsec compared with 95 nsec on VMware. Figure 10 shows that the respective distributions of the data are grouped together by the chosen sleep method.

5.2. CPU-bound Load

As in the idle scenario, with CPU load the distributions (figure 11) of `HPTSleep` miss as well as their medians are in the same range on all three platforms: Host OS has a 89.5 nsec median value of miss time (figure 11 (a)), QEMU virtualized OS has 104 nsec (figure 11 (b)), and VMware ESXi has 114 nsec (figure 11 (c)) respectively. Distributions of data with `uSleep` function are also close to each other: 54.7 µsec on Host OS, 61.3 µsec on QEMU virtualized OS and 56 µsec on VMware ESXi hypervisor-based OS. Considering the CCDF in figure 12, `HPTSleep` produces much longer tails than `uSleep`. Mean values are grouped together in the range between 0.1 and 1 msec while median values are around 100 nsec in case of `HPTSleep` and 50 µsec in case of `uSleep`. However, median values of `HPTSleep` differ by 15% max while medians of `uSleep` function differ up to 45%. Also the CCDF shows that outcomes of the `HPTSleep` function have a miss time overhead below 120 nsec with a 99.9% probability.
5.3. IO/Network-bound Load

The results of this experiment show that although medians of distributions sleep misses with \textit{HPTSleep} are significantly lower than with \textit{uSleep} function (tens of nanoseconds with \textit{HPTSleep} versus tens of microseconds with \textit{uSleep}) standard deviations of \textit{uSleep} distributions is one order of magnattude lower than \textit{HPTSleep}. All examined platforms exhibit this behavior. VMware ESXi shows a lower spread than other platforms in case of TSC counter (figure 13 (c)). QEMU has the highest spread with TSC counter within all other platforms (figure 13 (b)). The CCDF plot, in figure 14 shows that all mean values except QEMUs \textit{HPTSleep} mean are located near their medians. This outstanding behavior is caused by rare and powerful outbreaks which are in the same order of magnitude as worst case values for \textit{uSleep}. \textit{HPTSleep} function have \textit{miss time} value below in 99% cases.

5.4. Network-bound Load

Similarly to time acquisition measurements with network-bound load, the datasets presented in figures 15 and 16 also have "step-function-like" behavior. This behavior can be explained by fickleness of network data transmission, since the rmdt-protocol tries to take full advantage of available network and CPU resources which could be a cause of network congestions and packet loss that spawns fickleness. However, the CCDF representation (figure 16) is more similar to the IO/Network-bound case. With the system sleep call, mean values of the distribution are near the medians, at the same time in the case of \textit{HPTSleep} they are shifted to the right. With Network-bound load, \textit{HPTSleep} function has \textit{miss time} values below 120 nsec in 99% of cases.
6. Discussion of Results

In the preceding sections we described in detail the results of our timing measurements for each scenario. However, the cross comparison of distribution patterns for different load types, methods and VM models is required for specifying explicitly the impact of virtualization on timing.

6.1. Loads

CPU-bound load is characterized by very long outliers and as a consequence the mean values are shifted to the right relative to the medians. Median values under this load differ from the idle case only up to 15% in case of HPTSleep and up to 45% in case of uSleep function. Under IO/Network-bound load, the patterns of the timing distributions are similar to the idle scenario pattern. Medians, means and tail limits are very close to values in the idle scenario. Nevertheless, on VMware ESXi this load produces tails of distributions which are shorter than in non-virtualized measurements. Moreover, the median value of measurements with TSC counter on QEMU is lower than in the not virtualized scenario without any load. Pure network-bound load produces fickle datasets, due to the dynamic nature of data transmission. These distributions have the second longest tails after CPU-bound load. In this case distributions are strongly related on the virtual environment. This can be deducted from the tails behavior. They are grouped together by the system where measurements have been done. Considering worst-case execution time for applications running real-time, CPU-, IO-bound or network tasks, the difference in tails behavior should be taken into account.

6.2. HighPerTimer vs Standard Linux Calls

It's worth noting that in all scenarios, under every load the improved timing methods of the HighPerTimer library result in a significant increase in precision. Time acquisition operations without load have a lower overhead with direct access to hardware than system call: median values on 45% lower on Host OS, on 97% lower in case of QEMU, and on 47% lower on VMware ESXi. Moreover, using a standard system call for sleep operations can lead to even bigger performance degradations. The difference between the two methods can be up to 1000 times. This can be observed well on all CCDF plots, where distributions are grouped into two branches according to the applied operation.

6.3. Qemu vs VMware ESXi

Since hypervisor based model VMware ESXi has no unnecessary layer between hardware and VMM, a better timing precision was expected. In fact, comparing with QEMU virtualization the overhead of VMware ESXi on timing operations is lower. Nevertheless, it is also close to the overhead of the Host OS, however in case of time acquisition with system call, the difference between QEMU and VMware ESXi or Host OS can be up to 80%.
7. Conclusion
Our various measurement results for the case where one virtual instance is located on hardware allow to conclude the following: (i) While VMware ESXi has almost no basis overhead, QEMUs access to the hardware adds relatively high overhead (especially by introducing jitter when waking up from a sleep call); (ii) in case of sleep accuracy, the means of sleeps methods prevail over the virtualization environment effects; (iii) while considering the worst-case execution time, the virtualization used plays an almost neglecting role, the majority of values are measurably better on the host machine than (at least) on QEMU; (iv) the hypervisor-based approach brings clearly measurable benefits compared with the OS-based one.

Our ongoing work is to scale this experiment on the hosting scenario with several VMs and test under additional loads. This would allow us to demonstrate how the time hardware is shared among VMs and identify the resource limits.

Acknowledgment
This work has been funded by Volkswagen Foundation for trilateral partnership between scholars and scientists from Ukraine, Russia and Germany within the project CloudBDT: Algorithms and Methods for Big Data Transport in Cloud Environments.

References
[1] García-Valls M, Cucinotta, T , Lu, C, 2014. Challenges in real-time virtualization and predictable cloud computing. Journal of Systems Architecture 60 pp 726 – 40.
[2] I. Fedotova and E. Siemens 2014 System Time Issues for the ARM Cortex A8 Processor Proc. of the International Conference on Applied Innovations in IT (Köthen) p 7
[3] Intel 2016 Intel 64 and IA-32 Architectures Software Developer's Manual vol. 2
[4] G. Paoloni 2010 How to benchmark code execution times on Intel IA-32 and IA-64 instruction set architectures (Intel Corporation) vol 123
[5] Rosenblum, M Garfinkel, T 2005 Virtual machine monitors: current technology and future trends Computer 38 pp 39 – 47.
[6] P. Barham et al., Xen and the Art of Virtualization in Proc. of the 19th ACM Symposium on OS Principles (New York) pp 164-77
[7] Wang, G, Ng, T S E, 2010 The Impact of Virtualization on Network Performance of Amazon EC2 Data Center (San Diego: IEEE)
[8] Langer S G, French, T, 2011. Virtual Machine Performance Benchmarking. Journal of Digital Imaging, 24, pp.883 – 9.
[9] Adamczyk B, Chydzinski A., 2015. Achieving High Resolution Timer Events in Virtualized Environment I. McLoughlin, ed. PLOS ONE 10 p.e0130887.
[10] R. Uhlig 2005 Intel virtualization technology Computer 38 pp 48–56.
[11] Cerqueira F and Brandenburg B 2013 A Comparison of Scheduling Latency in Linux, PREEMPT-RT, and LITMUSRT in 9th Annual Workshop on OSs Platforms for Embedded Real-Time Applications (Paris) pp 19-29
[12] Litayem N, Ben Saoud S, 2011. Impact of the Linux Real-time Enhancements on the System Performances for Multi-core Intel Architectures. International Journal of Computer Applications, 17 pp 17 – 23.
[13] Bakharev A V, Siemens E, Shuvalov V P 2014 Analysis of performance issues in point-to-multipoint data transport for big data 2014 12th International Conference on Actual Problems of Electronics Instrument Engineering (Novosibirsk: IEEE) p 431
[14] Fedotova I Siemens E H Hu 2013 A high-precision time handling library Journal of Communication and Computer 10 pp 1076-86