A Quantitative Analysis of the Xen Virtualization Overhead

Fabricio Benevenuto, Cesar Fernandes, Matheus Santos, Virgilio Almeida, Jussara Almeida
Computer Science Department
Federal University of Minas Gerais
Brazil
{fabricio, cesar, mtcs, virgilio, jussara}@dcc.ufmg.br

Abstract

Recently, consolidation on virtual servers has gone from a small market to a massive industry migration which is affecting businesses across different kinds of companies. Virtualization can be useful to help IT organizations to reduce complexity and costs with management, ownership, and energy consumption, providing a flexible environment for the execution of IT services. In this context, understanding the main properties of the virtualization overhead and also quantifying this overhead is crucial for the deployment of applications in a virtual environment. In this work, we present a performance characterization of applications running on the Xen environment, aiming at identifying the most important aspects of the overhead imposed on applications by the virtual system. We use four benchmarks which stress different aspects of the virtual system. We compare the performance of the benchmarks on a single virtual machine (VM) against a Linux system and study the performance interference among VMs scaling on the same hardware platform. Our results explore the causes of this extra overhead and show that it can be significant.

1 Introduction

Recently, consolidation on virtual servers has gone from a small market to a massive industry migration which is affecting businesses across different kinds of companies. In fact, virtualization can help IT organizations to reduce complexity and costs with management, ownership, and energy consumption, providing a flexible environment for the execution of IT services [9]. The emergence of new virtual systems such as Xen [6], Denali [23], and VMWare ESX [8], and the development of hardware support for virtualization from Intel [13] and AMD [4] are some of the driving forces leading virtualization toward production systems and Web services.

Depending on the virtualization technique, the complexity and cost of virtualization can change drastically. As an example, in systems based on pure virtualization, in which guest VMs run on the top of a host operating system (OS) without any awareness of the underlying virtualization layer, the virtualization overhead is much higher than on systems based on para-virtualization, which makes the guest OS aware of virtualization.

Our main goal with this work is to understand the main properties of the virtualization overhead through a performance evaluation of applications when they are migrated to Xen, a free and open source virtual system which adopts para-virtualization. We use four benchmarks which stress different aspects of the virtual system and monitor how the main architectural aspects, such as number of instructions retired, cycles per instruction, cache and TLB misses are affected when we move the benchmarks from Linux to Xen. Moreover, we study how these metrics are changed as we scale virtual machines on Xen. Our results identify the most important causes of this overhead and show that it can be significant.

The rest of this paper is organized as follows. Next Section presents related work and Section 3 discusses aspects of Xen as a necessary background, and also describes the experimental environment, tools and benchmarks used in the experiments. In Section 4, we present a performance study of the overhead that the Xen environment imposes on applications, and the performance degradation that occurs when there are VMs sharing the same CPU. Finally, Section 5 concludes the paper and presents directions for future work.

2 Related Work

Most part of the resurgence of interest on virtualization is due to the possibility of deployment of applications on VMs with low performance overhead. There are several works which compares the performance of virtualized applications at different virtualization platforms and also with non-virtual system [6, 7, 12, 15, 17, 23]. In common, these studies focus on analysis of the overall application performance (i.e. measuring system response time or providing a benchmark result) on the virtualized system. In our work, we compare the performance impact of the Xen virtual environment on different benchmarks, focusing on capturing the most important characteristics of the Xen virtualization overhead.

There are only a limited number of studies which focuses on understanding the characteristics of the virtual environment overhead [5, 20]. The former work presents Xenoprof, a system-wide statistical profiling tool able to sample hardware events such as instructions processed, busy clock cycles, hits, and misses on TLB and processor caches. The Xenoprof work provides a case study of network applications in order to show how Xenoprof can be used to diagnose bugs or performance problems on Xen. The second work uses Xenoprof to study performance issues of a CPU intensive application (the SPECjbb2005) running on a single VM. Our work also uses Xenoprof, but it explores the differences among different benchmarks, showing how different workloads can cause different levels of interference among VMs sharing the same hardware platform. As a side effect of our experimental analysis, we present a performance evaluation of Xenoprof overhead, which was not provided in [20].

3 Methodology and Environment

This Section describes Xen, the virtual environment used in our analysis, the experimental environment, the benchmarks and tools used in our experimental evaluation. Next Section presents our results.

3.1 Xen Architecture

Xen is a free and open-source virtual machine monitor (VMM) which allows multiple (guest) operating system (OS) instances to run concurrently on a single physical machine. It uses para-virtualization, where the VMM exposes a virtual machine abstraction slightly different from the underlying hardware.

The Xen system has multiple layers. Figure 3.1 provides an overview of Xen architecture. The lowest and most privileged is called VMM. Each guest OS runs on a separate virtual machine called domain or guest VM. The VMs are scheduled by the VMM to make effective use of the available physical CPUs. Each application, running on a guest OS, accesses hardware devices via a special privileged virtual machine called *isolated driver domain (IDD)*, which owns specific hardware devices and run their I/O device drivers. All other domains (*guest domains* in Xen terminology) run a simple device driver which communicates with the driver domain to access the real hardware devices. The IDD can access directly the hardware devices it owns. However, a guest domain accesses the hardware devices indirectly through a virtual device connected to the IDD. The IDD maps through bridges or routing the physical interface to its virtual interface which communicates with the guest virtual interface. The guest domain exchanges requests and responses with the IDD over an I/O channel. In order to avoid copying, references to page-sized buffers are transferred over the I/O channel instead of the actual I/O data [11].

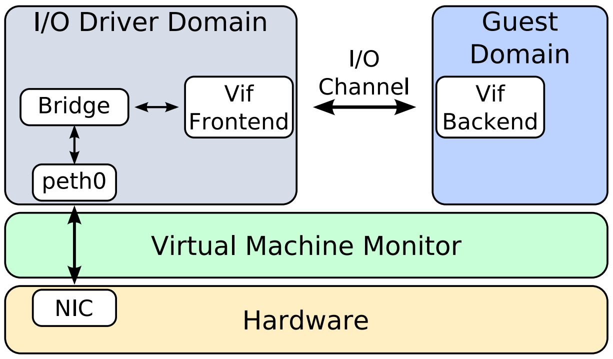


Figure 1. Overview of Xen Architecture

In order to improve the I/O mechanism of virtualization systems, Intel is currently developing a new hardware able to provide to VMs direct access to hardware I/O operations [14]. We believe that the IDD overhead might be reduced or even eliminated for machines with this kind of hardware support.

3.2 Workloads and Benchmarks

In order to compare performance of applications running on Linux and Xen, we need to use benchmarks that, each of them, execute the same task on any system. [1]. In this context, we choose as benchmarks real applications which stress different parts of the system. Particularly, the workload used in our experiments is generated by the following benchmarks:

kernel compilation: The compilation of the source code of a kernel consists of a CPU intensive application, which calls several functions, stressing system CPU. The kernel version used is 2.6.12.

Web Server (static): We use a Web server providing only static content (HTML files) in order to evaluate its execution on VMs. We use httperf [21] as clients and Apache [1] version 2.0.55 as the Web server. The httperf is a tool which allows generating several HTTP workloads and measuring the performance of the Web server from the point of view of the clients. We run httperf on a client machine, sending requests to the server, measuring throughput and server response time of the requests. The workload used is generated by SPECWeb99 [19] and it does not contain dynamic content. The same workload was used in [10].

Web Server (dynamic): In order to evaluate a Web server providing dynamic content we developed the part of search of an e-commerce server. Our application was developed in PHP and it access a database (mysql version 5.0) of a bookstore characterized and detailed described in [18]. As we did for the static Web server, we use httperf as clients and Apache hosting the e-commerce server. The workload used to feed httperf is composed of only search operations obtained from logs of a real e-commerce server also described in [18]. For all the experiments with this benchmark we configured the database and the e-commerce server on the same VM in order to provide a fair comparison with the other benchmarks.

Disk: For disk intensive test, we chose not to write our, but rather to use a copy of a file. Particularly, for all experiments, we execute on a guest VM a copy of a 2 GB file to the same VM partition.

[1]A standard benchmark for virtualization is under elaboration by SPEC [22]

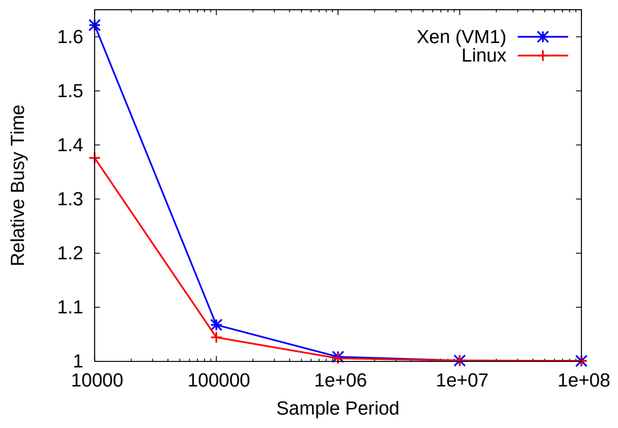


Figure 2. Xenoprof and Oprofile Overhead

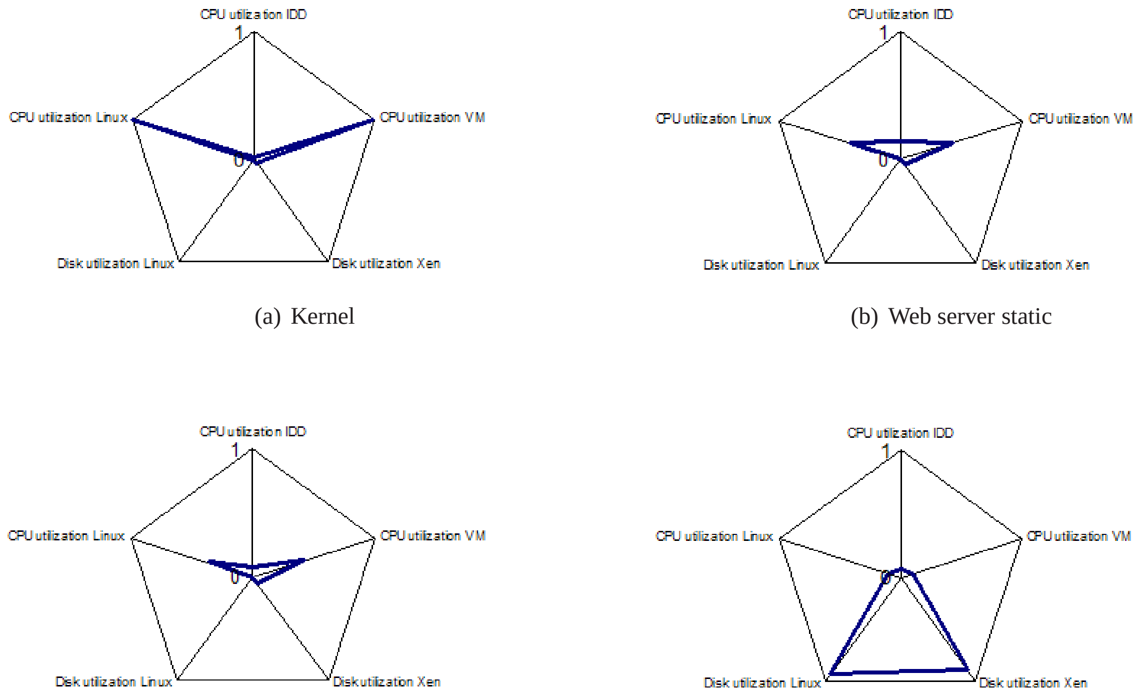
3.3 Monitoring Framework

In order to provide a performance evaluation of virtualized applications, we need to be able to measure the virtualized system. As part of our monitoring framework, we developed an application, which we call *Xencpu*, to measure CPU busy time on Xen. This tool is based on the source code of *xm top* tool, provided with Xen, and was designed aiming at the automatic execution of our scripts. The CPU busy time on the Linux system, the number of packets, and processor interrupts on both environments were obtained from */proc* directory. In order to measure system metrics such as number of instructions retired, TLB and processor cache misses, we use Xenoprof [20], a system-wide statistical profiling tool for Xen, which collects periodic samples of performance data. The Xenoprof code is based on a Linux tool called Oprofile [2] which have the same functionality as Xenoprof for machines running Linux. To measure the same metrics on Linux, we use Oprofile.

In order to provide a fair comparison between Xen running Xenoprof and Linux running Oprofile we define a sample period in which Xenoprof and Oprofile do not cause a high overhead on the system. Figure 2 compares the CPU busy time to execute the kernel benchmark for a machine Linux and running Oprofile and the same machine using Xen and running Xenoprof. For Linux, we plot the values obtained running Oprofile relative to the system without running Oprofile. For the VM environment, we plot the values running Xenoprof relative to the VM executing the same benchmark running with Xenoprof. For small sample periods both, Xen and Linux starts to enter in a trashing state due the high number of interruptions caused by Oprofile and Xenoprof. The trashing effect is more perceptive in Xen, since there are more instructions being executed in this system due to the virtualization. However, for a higher sample period both tools do not cause significant interference in the system performance. For the kernel benchmark we chose 10^7 as the sample period for event instructions retired. We performed a similar study for each benchmark used and event analyzed. Figure 2 is also useful to provide an understanding of Xenoprof overhead, since this kind of analysis was not presented previously [20].

3.4 Experimental Setup

For all experiments, we use a 64-bit two-CPU 3.2 GHz Intel Xeon server, with 2 GB of RAM, one disk with 7200 RPM and 8 MB of L2 cache, and two Broadcom Realtek Gigabit Ethernet card. The server is directly connected to a single-CPU AMD Athlon64 3 GHz, with 2 GB of RAM, and two Realtek Gigabit Ethernet card. We configure Xen to use i386 architecture. We use Xen version 3.0.4 and the VMs runs XenoLinux derived from Linux Debian Etch i386 distribution, kernel 2.6.16. The client machine also uses the same Linux distribution and



(a) Kernel (b) Web server static

(c) Web server dynamic (d) Disk

Figure 3. Kiviat representation for each benchmark

kernel. The Linux server is configured with 1024 MB of RAM, and the single VM is configured with 512 MB of RAM as well as the IDD. For experiments that increase the number of allocated VMs, each one uses 256 MB of RAM. The credit scheduler [3] is used as the Xen scheduler, but we do not limit the amount of CPU resources for the VMs.

4 Performance Evaluation

This Section presents an experimental performance study of applications running on the Xen VMM. We configure the virtual environment with one IDD and one guest, each one using a different CPU. The Linux system uses a single CPU.

4.1 Overheads of Different Benchmarks

In order to demonstrate how different applications are affected by the Xen virtualization overhead, we compare the four benchmarks described in Section 3.2 executing on Linux and Xen systems. Figure 3 shows four Kiviat representations which compares CPU and disk utilization on Linux and Xen systems for each benchmark. In a Kiviat graph each radial line starting at the central point 0, represents one metric with maximum value 1 [16]. We plot five metrics on this graph, namely, the CPU utilization for the Linux CPU, for the IDD and for the VM, and disk utilization on both systems. Each graph represents a benchmark. For the Web server benchmarks we choose 600 requests/s and 7 requests/s as request rates for static and dynamic content, respectively.

For the kernel compilation benchmark, the CPU utilization is almost 1 for both, Linux and the guest VM.

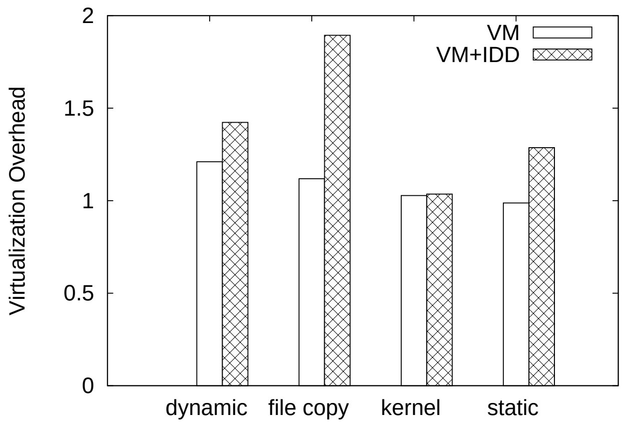


Figure 4. Overhead of a single VM

Since this benchmark does not execute a representative number of I/O operations, the CPU utilization on the IDD is almost 0. Observing the Web server which provides only static content, we can observe the VM and the Linux system running competitively and also note a high CPU processing on the IDD due to I/O operations. The same behavior can be observed for the dynamic Web server. The disk activity is negligible for both Web server benchmarks executing on Linux, but there exists a disk utilization of 4% and 6% respectively for static and dynamic Web servers. There is a high CPU activity on the IDD for these two Web benchmarks due to the processing of network I/O operations. Also, observing the disk benchmark, we can note that the disk I/O activity is responsible for 7% of CPU utilization on the IDD.

4.2 Overhead of a Single VM

In order to decompose the main architectural factors that change on a VM environment, we return to the classic equation of CPU busy time (eq. 1), which depends upon three characteristics: number of instructions of an application (IC), clock cycles per instruction (CPI), and the clock rate.

$$CPU\ busy\ time = IC \cdot CPI \cdot Clock\ rate \tag{1}$$

The clock rate is not interesting to the focus of these analysis since we are dealing with the same hardware platform. In fact, we are interested in understanding how IC and CPI change from Linux to Xen and also understand the causes of their increases.

Firstly, we analyze the virtualization overhead for each benchmark. Figure 4 provides a comparison of busy time measured on the virtual environment to execute each of the four benchmarks relative to the busy time measured on Linux. We can see that the virtualization overhead (VM + IDD) for the file copy is the highest, followed by the dynamic and static Web server benchmarks, and then by the kernel compilation. We next, explore the reasons for these differences.

We compare several metrics measured on Linux and on a single VM, also measuring the analyzed events on the IDD. Figure 5 shows, for all analyzed benchmarks, the instruction count (IC), the CPU cycles per instruction (CPI), the number of misses on the caches L1 and L2, and the number of misses on the TLB of instruction and data. These numbers are all relative to the execution of the benchmarks on the baseline Linux system.

Observing the figure of IC, note that, the difference between each pair of two columns represents the amount of instructions executed by the IDD, which is responsible for intermediating the hardware access for the VMs. For the kernel compilation benchmark, the amount of instructions executed by the IDD is very small since this

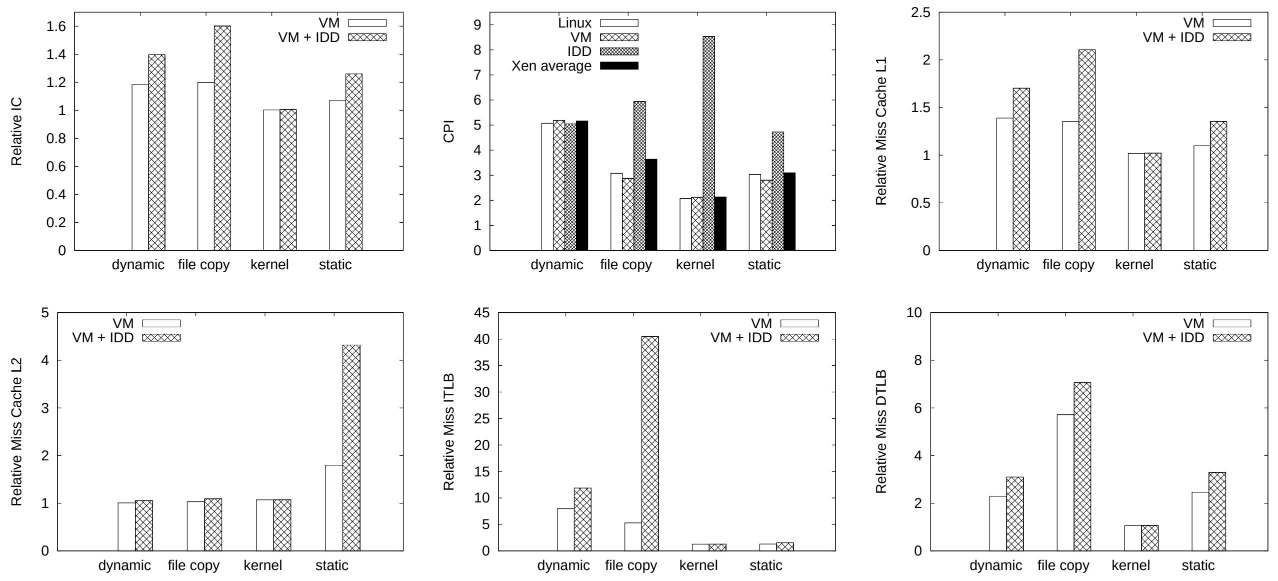


Figure 5. Benchmarks running on a Xen VM relative to Linux: IC, CPI, and L1 cache misses (top), L2 cache misses, instruction TLB misses, and data TLB misses (bottom)

benchmark requires few disk and none network I/O operations. However, for the other benchmarks there are more work executed by the IDD compared to the kernel benchmark. As an example, the IDD is responsible for 29% of the total number of instruction for the file copy whereas for the kernel benchmark the IDD is responsible for less than 0.2%.

We now, turn our attentions to what happens with the cycles per instruction of both Xen components compared with Linux. In order to compute the CPI we divided the number of busy cycles to execute the benchmark by the number of instructions executed. The number of busy cycles was calculated multiplying the processor frequency per the CPU busy time. The average CPI on the Xen environment was obtained as the mean of the IDD and VM CPI, weighted by their respective IC. The dynamic benchmark makes several access to a database installed inside the VM, which are in great part operations with high CPI. Compared with the other benchmarks, the IDD CPI of the dynamic Web server is still high (ex. higher than the static Web server benchmark) and the VM CPI is the highest one, given for this benchmark the first place for the average CPI. The highest IDD CPI is for the kernel benchmark, almost 8.6 cycles per instructions. However, since the IC on the IDD is insignificant for the kernel compilation, the average CPI is not affected by the IDD CPI, and thus, it is very close to the VM CPI.

4.3 Interference Scaling VMs

This Section analyzes the interference, I, among VMs when they share the same CPU. Decomposing the main factors which change when we increase the number of VMs sharing the same CPU, we name *Instruction Growth* (IG) and *Instruction Dilation* (ID) any increase on the IC and CPI, respectively, on a single VM due to other VMs concurrently running on its same hardware platform. We next quantify and decompose I into IG and ID for the benchmarks analyzed, understanding the impact of the virtualization platform on each benchmark.

Both, IG and ID are responsible for interference when we run multiple VMs on the same CPU. In this context, our first step is to quantify I, IG and ID as we increase the number of VMs. We run one kernel compilation per VM. We measure interference as the busy time to execute the benchmark from the point of view of a single VM

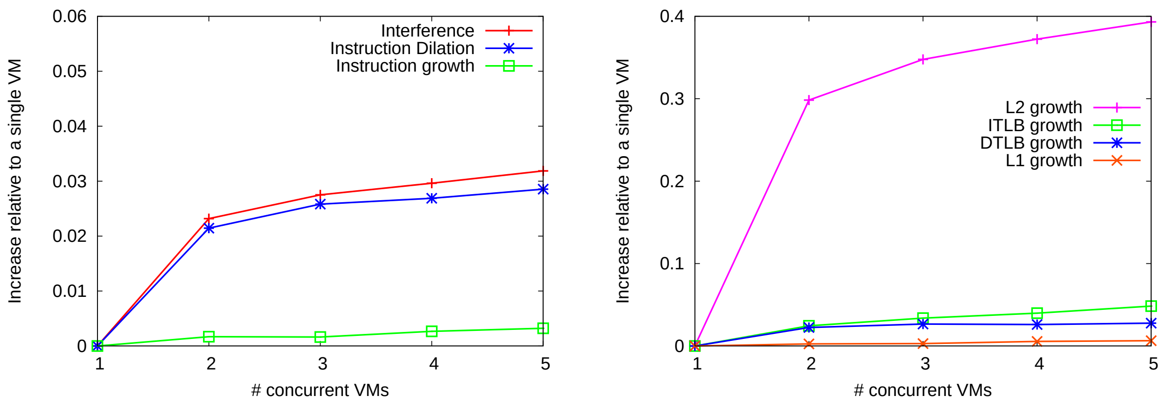


Figure 6. Performance study of interference and its causes with multiple VMs sharing the same CPU. I, ID and IG for the Kernel benchmark (left) and ID causes for the kernel benchmark (right)

when it is sharing the CPU with other VMs, divided by the busy time to execute the same benchmark on a single VM when it is not sharing the CPU. We also compute IG and ID relative to one single VM solely occupying the CPU. The virtual environment is configured with the IDD running on a separate CPU and the VMs sharing the other CPU. Figure 6 (left) shows I, ID and IG as a function of the number of VMs running the benchmark. There is a higher increase on I from 1 to 2 VMs and then, I starts to increase slowly reaching 3.19% for 5 VMs. The same behavior can be observed for ID, which is the main cause of interference. We can also observe a small increase on IG, less than 0.32% for 5 VMs.

Since the main cause of interference is ID, we analyze its causes. We plot level 1 (L1) and level 2 (L2) processor cache misses, and data (DTLB) and instruction (ITLB) TLB misses, all relative to one single VM. We assume L1 cache miss as the sum of the misses and hits on L2 cache, since L1 cache miss is the unique measured event which cannot be directly obtained from Xenoprof [20]. Figure 6 (right) shows these hardware events as a function of the number of VMs. Note that the L2 cache misses curve has a similar behavior as ID. These results show strong evidences that an increase on L2 cache misses is the most important reason to ID for this benchmark and, consequently, the most important cause of interference.

5 Conclusions and Future Work

This work presents a performance evaluation of applications running on the Xen environment, identifying the most important aspects of the overhead imposed on applications by the virtual system. Summarizing, our main findings are:

- The virtualization overhead can be significant, specially for applications which relies on I/O operations.
- In terms of the cause of virtualization overhead, the CPI observed for the IDD is higher than the VM CPI and thus, applications which use a significant amount of IDD resources may experiment a higher virtualization overhead due to an increase on the average CPI. Also, the IDD presents the highest miss ratio on caches L1 and L2 compared to the VM and Linux, justifying its high CPI.
- Interference due to VMs scaling on the same CPU is caused by instruction dilation ID, which is, in most part, caused by an increase on the number of L2 cache misses.

We believe that our findings can be used to guide the design of mechanisms able to diminish the cost of virtualization. More importantly, we showed that the virtualization overhead can increase when VMs are sharing the same resource. This extra overhead is usually not accounted on performance models or considered by scheduler parameters.

As future work we plan to extend our analysis for different configuration scenarios and benchmarks. Another interesting direction is to create a model able to predict the overhead of a workload for a large number of VMs.

Acknowledgments

This work was developed in collaboration with HP Brazil P&D.

References

[1] Apache. `http://httpd.apache.org`.
[2] Oprofile. `http://oprofile.sourceforge.net`.
[3] Xen Credit Scheduler. `http://wiki.xensource.com/xenwiki/CreditScheduler`.
[4] AMD. Pacifica technology. `http://enterprise.amd.com/us-en/AMD-Business.aspx`.
[5] P. Apparao, R. Iyer, and D. Newell. Architectural Characterization of VM Scaling on an SMP Machine. In *Proc. of XHPC, LNCS, 4331*, Sorrento, Italy, Dec 2006.
[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. of 19th ACM SOSP*, New York, NY, Oct 2003.
[7] L. Cherkasova and R. Gardner. Measuring CPU overhead for I/O Processing in the Xen Virtual Machine Monitor. In *Proc. of USENIX Annual Technical Conference*, Apr 2005.
[8] S. Devine, E. Bugnion, and M. Rosenblum. Virtualization system including a virtual machine monitor for a computer with a segmented architecture. Technical Report US Patent 6397242, vmware, Oct 1998.
[9] S. Eranian. Enhanced Virtualization on Intel Architecture-based Servers. Technical Report Intel Solutions White Paper, Intel, Jul 2005.
[10] F. Benevenuto and C. Teixeira and M. Caldas and V. Almeida and J. Almeida and J. R. Santos and G. Janakiraman. Performance Models for Applications on Xen. In *Proc. of XHPC, LNCS, 4331*, Sorrento, Italy, Dec 2006.
[11] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe Hardware Access with the Xen Virtual Machine Monitor. In *Proc. of OASIS*, Oct 2004.
[12] D. Gupta, R. Gardner, and L. Cherkasova. XenMon: QoS Monitoring and Performance Profiling Tool. Technical Report HPL-2005-187, HP Labs, October 2005.
[13] Intel. Vanderpool technology. `http://www.intel.com/technology/computing/vptech/`.
[14] Intel. Virtualization Technology for Directed I/O. `http://www.intel.com/technology/itj/2006/v10i3/2-io/5-platform-hardware-support.htm`.
[15] J. Matthews and W. Hu and M. Hapuarachchi and T. Deshane and D. Dimatos and G. Hamilton and M. McCabe. Quantifying the Performance Isolation Properties of Virtualization Systems. In *Proc. of First USENIX ECS*, San Diego, CA, Jun 2007.
[16] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, INC, 1st edition, 1991.
[17] D. Magenheimer and T. Christian. vBlades: Optimized paravirtualization for the Itanium processor family. In *Proc. of USENIX VM*, May 2004.
[18] D. Menasce, V. Almeida, R. Riedi, F. Peligrinelli, R. Fonseca, and W. M. Jr. In Search of Invariants for E-Business Workloads. In *Proc. of the 2nd ACM e-Commerce Conference*, pages 56–65, Minneapolis, MN, Oct 2000.
[19] D. Menasce, L. Dowdy, and V. Almeida. *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.
[20] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel. Diagnosing Performance Overheads in the Xen Virtual Machine Environment. In *1st ACM/USENIX VEE*, Chicago, IL, June 2005.
[21] D. Mosberger and T. Jin. httperf: A Tool for Measuring Web Server Performance. In *Proc. of WISP*, pages 59—67, Madison, WI, June 1998.
[22] Standard Performance Evaluation Corporation. http://www.spec.org.
[23] A. Whitaker, M. Shaw, and S. Gribble. Scale and Performance in the Denali isolation kernel. In *Proc. of OSDI*, Boston, MA, Dec 2002.