Virtual Systems Tracing for Performance Analysis

Parisa Heidari, Mathieu Desnoyers and Michel R. Dagenais*

Department of Computer and Software Engineering, Ecole Polytechnique de Montreal, P.O. Box 6079, Succ. Downtown, Montreal, Quebec, H3C 3A7, Canada

Abstract: This paper describes extensions to the Linux Trace Toolkit next generation (LTTng), to trace the Xen hypervisor for efficiently tracing complete systems. LTTng is a low impact, modular, extensible, operating system tracer, while Xen is an hypervisor based on paravirtualization. Typically, one or several instances of Linux will run on top of the Xen hypervisor. The extended LTTng is able to trace all Linux instances and the hypervisor layer to give a global view of the system.

An automated benchmark was created to measure the overhead imposed by tracing and Xen paravirtualization. It compares the performance of systems, with and without virtualization and tracing. The results obtained with different tests, using this procedure, are presented and show a negligible impact caused by tracing and a reasonable overhead caused by paravirtualization. Nonetheless, for some I/O intensive applications, the overhead imposed by paravirtualization is less negligible.

Keywords: Tracing, virtualization, performance analysis, Xen, Linux.

1. INTRODUCTION

With the proliferation of online services, server consolidation through virtualization is increasingly popular. Among the different virtualization solutions available, Xen is particularly interesting being open-source and low overhead. Xen is based on paravirtualization, where the operating system is ported to a software interface exported by the hypervisor. Typically, one or several instances of Linux ported to Xen will run on top of the Xen hypervisor.

Virtual machines have been used for more than thirty years to consolidate server applications, share existing resources and reduce costs [1]. In many cases, the powerful servers now available can adequately support several different applications, each possibly running in a different virtual machine. Various research prototypes and commercial products have been proposed for virtualization [2].

In virtualization, a software layer called 'Virtual Machine Monitor' (VMM) manages shared resources. In Xen, the VMM is called hypervisor as it is located under the operating system (supervisor), and thus is more privileged [3]. Earlier systems usually emulated in software the virtual hardware on which to run the legacy operating systems. Emulation of resources is usually slower and in many cases difficult to implement. A more recent technique is paravirtualization [3]. In paravirtualization, the guest operating system is slightly modified to send hardware requests to the VMM instead of to emulated hardware.

*Address correspondence to this author at the Department of Computer and Software Engineering, Ecole Polytechnique de Montreal, P.O. Box 6079, Succ. Downtown, Montreal, Quebec, H3C 3A7, Canada; Tel: +1(514) 340-4711, Ext. 4029; Fax: +1(514) 340 5139;
E-mail: michel.dagenais@polymtl.ca

Xen [3] is an interesting open source product offered under the GPL license. Although Xen supports full-virtualization as well as paravirtualization, this paper focuses on paravirtualization and modified Linux guest operating systems. In a typical setup, one or more virtual machines (DomainU) are created, depending on the needs, and a privileged domain (Domain0) directly accesses the physical devices. Hardware requests coming from DomainU virtual machines are received by the VMM, called the 'hypervisor' in Xen, and answered by Domain0. This indirection, through the hypervisor and Domain0, adds a delay which could be problematic in some real-time applications [4].

The focus of the work presented here is low overhead detailed tracing of Xen and Linux systems. If simple profiling can efficiently provide a clear picture of the CPU usage in a program, detailed tracing is often required in order to really understand and characterize the performance of complex interactions between several applications and the operating system [5, 6]. Furthermore, when two layers are involved underneath the applications, (the operating system and the hypervisor), complete combined tracing in each virtual machine and in the hypervisor becomes even more valuable.

The objective of this research is to extend operating system tracing to support virtualization and then trace events in all layers from hypervisors, such as Xen, to the Linux operating system, running on both real and virtual systems. The combined information of events collected at the different levels will offer a global picture of the system behavior. Our first contribution is to extend the Linux Trace Toolkit Next Generation (LTTng)[1] to support Xen and be a suitable alternative to Xentrace. The second contribution is

[1] http://ltt.polymtl.ca

1874-110X/11

the development of an automated performance benchmark which runs various realistic tasks under different configuration scenarios, in order to measure the tracing overhead. The overhead of each component and each alternative is evaluated under different configurations. The third contribution is the performance evaluation, conducted using the extended LTTng and the automated performance testing procedure.

2. TRACING THE OPERATING SYSTEM

During recent years, tracing has been successfully used to understand interactions between processes and the operating system by instrumenting system calls. A lot of production or execution problems have been solved using tracers. A tracing tool adds, statically or dynamically, some trace points or probes called 'instrumentation'. Dynamic instrumentation is adding trace points while the operating system is running. Static instrumentation is adding the trace points directly in the code before execution, and requires the kernel to be recompiled after the modification.

Among the various tracers available, SystemTap [7], Dynamic Tracing (DTrace) [8], and Linux Trace Toolkit (LTT) [9] should be mentioned. Although they all offer similar functionality, they differ in their instrumentation mechanism and, as a consequence, in performance overhead on the operating system.

2.1. Linux Trace Toolkit Next Generation

Linux Trace Toolkit Next Generation (LTTng) [10] is based on static instrumentation. Trace points and their associated handlers are defined as a function call to the corresponding handler inserted at suitable locations in kernel code. The modified kernel is then recompiled. Once a trace point is hit, a function call is executed and the CPU jumps to the associated handler. Results have shown that the impact of the branch mechanism is less than the impact of a software interrupt trap used by most dynamic tracers. Tests results and discussions could be found in the Linux Kernel Mailing List [11].

In comparison with the earlier LTT, LTTng brings several technical improvements such as increasing time stamp accuracy, high speed user space tracing, atomic operations to have a real lock-less mechanism, tracing non-maskable interrupts (NMI), and markers. The latter is introduced below.

LTTng proposes kernel markers, a source code annotation to define trace point locations and arguments (data to log or to be accessed by probe handlers). Depending on being active or not, markers will be replaced with a function call or a 'jump' respectively. The idea is to have kernel markers inserted at relevant locations in the kernel by the code authors, in many cases instead of printk statements (the print instructions specifically used in Linux kernel). These markers can then be used by any tracing tool, including LTTng. This way, key events in the kernel are identified by predefined trace points identified by markers.

3. TRACING AND VIRTUALIZATION

Extending from a physical machine, with one operating system, to a virtual system, with multiple layers and different operating systems, adds new requirements to tracers and renders existing tracers incapable of representing time-coherent information across layers [12]. Xen provides its own tracer, Xentrace. It works fairly well in isolation but has not been interfaced to a tracing system which could simultaneously present the events from the operating systems and the applications executing in the virtual machines. A primitive solution is to trace virtual domains with existing tracers prepared for ordinary operating systems, and trace the hypervisor with Xentrace. However, considering the close interactions between domain0 and the hypervisors, events happening in these two layers should be measured and recorded through the same mechanism and time base.

The idea is to extend a tracer which is able to give a detailed picture of an operating system, in order to have a detailed global picture of all layers and virtual machines.

3.1. Tracing Virtual Domains

There is no particular difficulty in tracing a virtual machine rather than a physical one. In each domain, LTTng patches may be applied to the Linux kernel. Since both Xen and LTTng are currently distributed as patches to the official kernel, care is advised to avoid conflicts between the modifications required for each system.

3.2. Tracing the Hypervisor

In order to trace Xen with LTTng, some trace points are defined, inspired from the trace points of Xentrace. Thus, the number and location of trace points is identical, simplifying their performance comparison. The existing Xentrace calls were replaced with calls to macros that can be remapped to either Xentrace or LTTng. Additional header files are defined in which there is a conditional compilation definition to activate or not LTTng.

Xen supports the dynamic allocation of VCPUs and CPU hotplug. Hot plugging provides the ability to manage devices being attached to or removed from a running system. In a real machine supporting hotplug, the number of CPUs may change while the system is running. In Xen virtual systems, the number of VCPUs, virtual CPUs assigned to a virtual machine, may similarly vary while the virtual system is running. Therefore, the number of CPUs in each domain may vary during a trace. On the other hand, LTTng uses per CPU buffers. Thus, LTTng and the underlying RelayFS had to be extended to allocate new per CPU buffers when CPU hotplug events are encountered. The LTTng daemon, lttd, needs to use inotify in order to be notified of new tracing channels. Inotify is an operating system facility which provides a notification service whenever a given event happens.

3.3. Global Image of Virtual Systems

After porting LTTng to Xen-compatible operating systems and the hypervisor, it is able to give an interesting global view of virtual systems including all its layers and domains. Running a compression application on domain0 and similarly on domainU, a trace was taken on domain0, domainU and the hypervisor. Traces were visualized with

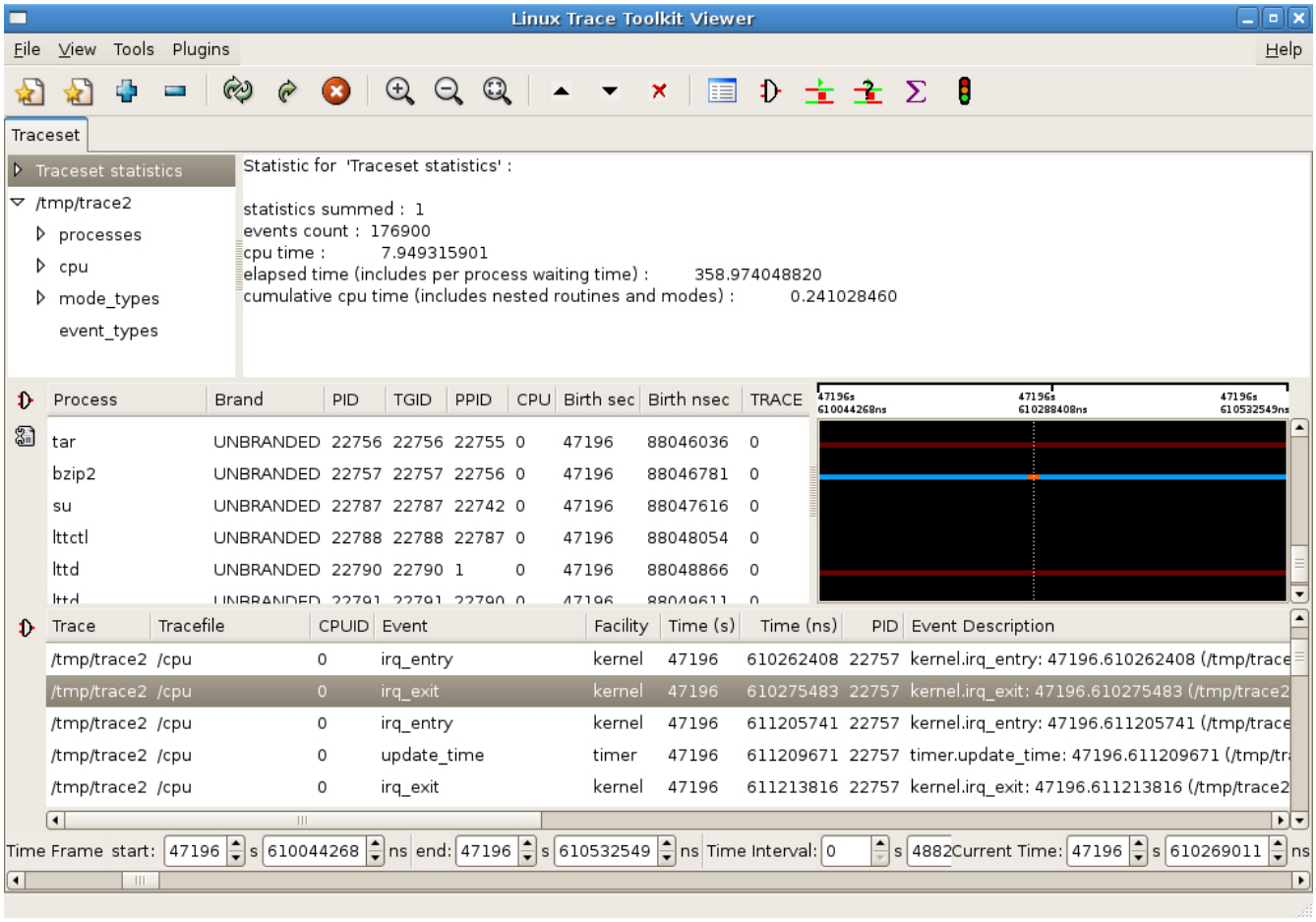

Fig. (1). Tracing Xen-domain0.

the LTTng Visualizer, LTTV. LTTV is able to visualize multiple traces from different domains and layers in the same window. Figs. (**1**, **2**) represent traces taken in domain0 and domainU, respectively. Fig. (**3**) represents the traces taken in domain0 and the hypervisor concurrently. The latter could be very useful to give an idea about the interactions of domain0 and the hypervisor.

4. AN AUTOMATED BENCHMARK

An automated benchmark was built. Although these benchmarking scripts could be used for many different types of tests, we only present here the particular setup used to test virtual machines tracing, the focus of this paper.

Three scenarios were studied to better characterize the performance of virtualization and tracing. The first one looks at the impact caused by Xen. Linux domain0, running over Xen, but having access to real resources, is compared with a physical machine running an unmodified Linux kernel. In the second scenario, the performance of a virtual machine, Linux domainU, is compared with that of a physical machine. In the third scenario, the impact caused by LTTng is observed on domain0, domainU and a real machine in four possible configurations: LTTng is not compiled, compiled with markers disabled, active in flight recorder mode (the trace information is written into the buffers without being recorded to disk), and finally full tracing with data recorded to disk. As a complementary test, the impact of LTTng-Xen (the module tracing the hypervisor) and Xentrace are also compared.

Besides measuring the performance for the mentioned scenarios, each test is repeated a number of times to measure the impact of caching, mostly having disk files resident in the memory buffer cache and thus saving on disk accesses. A second complementary test studies the impact of using virtual disks (loopback files) instead of physical disk partitions. Previous tests have shown that using loopback files in a normal machine can lightly degrade the performance [13]. Besides considering the four aforementioned configurations of LTTng on a physical machine, on a privileged guest and on a domain-U (both running under Xen with physical partitions), all measurements have been repeated for a virtual machine running on a virtual disk. Tracing of the hypervisor layer by both Xentrace and LTTng-Xen is compared. Each complete test contains 18 reboot steps, each representing a different kernel and machine configuration.

An automated mechanism is implemented in order to perform all these measurements. Measuring various parameters on different kernels and machines is particularly tricky and time consuming. For this purpose, an automated benchmark was designed to be easily repeatable and self

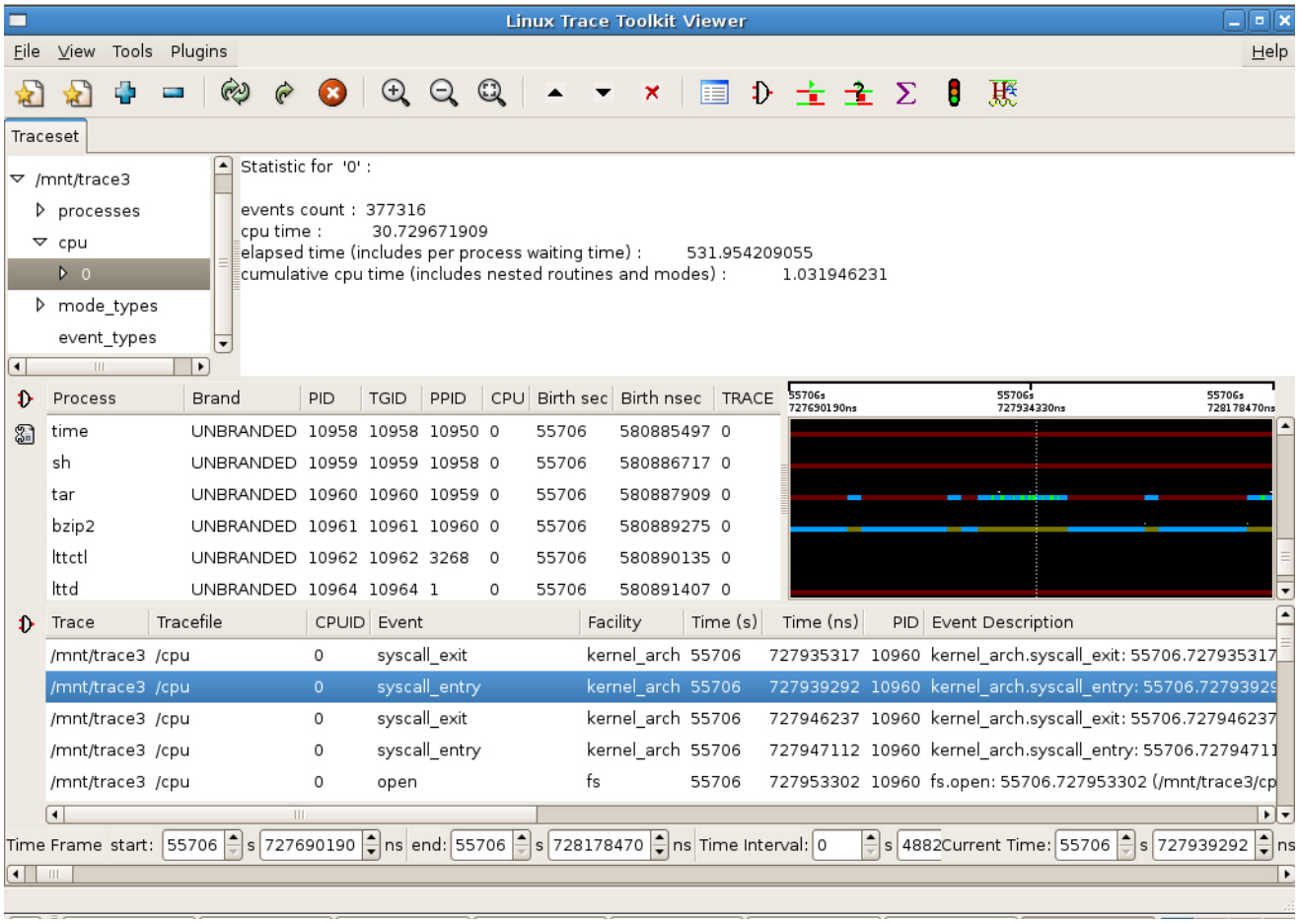

Fig. (2). Tracing Xen-domainU.

contained; it comes with all the tools and kernel variants required. It is extensible to test various performance characteristics and different configurations of real or virtual machines.

In this study, we have used the following setup. For each test (e.g. kernel compilation, directory archiving, file compression…) the only thing a user needs to do is choose the test and run the benchmark. It starts from the first iteration (step 0), executes the test, reboots the system with the next kernel, repeats the same test, continues with the next kernel configuration, creates a virtual domain if necessary and goes on until the last iteration (step 17). A result file is created in the result directory for each step. Each iteration tests a different configuration. Pseudo code of Fig. (**4**) clarifies the concept.

The benchmark consists of a text file describing all configurations, an executable script to start the test and define appropriate settings, an initial script located in '/etc/init.d', a daemon (main script) running in the background, a script file for each test type and a short text file saving the current iteration (step number) across reboots. Most parameters are easily modified in the configuration file. The starter script prepares the system for the test, determines the appropriate kernel for the first step and reboots the system. Then, the initial script in '/etc/init.d' executes. It has been given the lowest priority to make sure that all services are up and running when it starts.

After starting the test, an additional 2 minutes sleep is requested to make sure that all other services have finished their initialization and the system is idle. Before starting the test, an archiving application runs in order to fill empty buffers. After running this application, although cache buffers are not empty, no reusable content exists in the buffers (cache cold). Finally, it executes the test running the corresponding script. This procedure is easily extensible.

Thus, the contribution brought by this test procedure has been to provide a benchmark which automatically configures a test system, including virtual and physical machines, based on user's settings, before running a number of regular benchmarks on the specified configurations.

5. CONFIGURATION AND APPLIED TESTS

The tests may be divided into two categories, real applications stressing particular parts of the system, (compilation, archiving and compression), and standard benchmarks simulating different types of load, (Dbench, SPECviewperf9.1 and LMbench). The machine performing the test is an Intel Pentium 4, 3GHz hyperthreaded processor, with 2 GB RAM, and a single 320 GB 7200 RPM SATA SEAGATE.

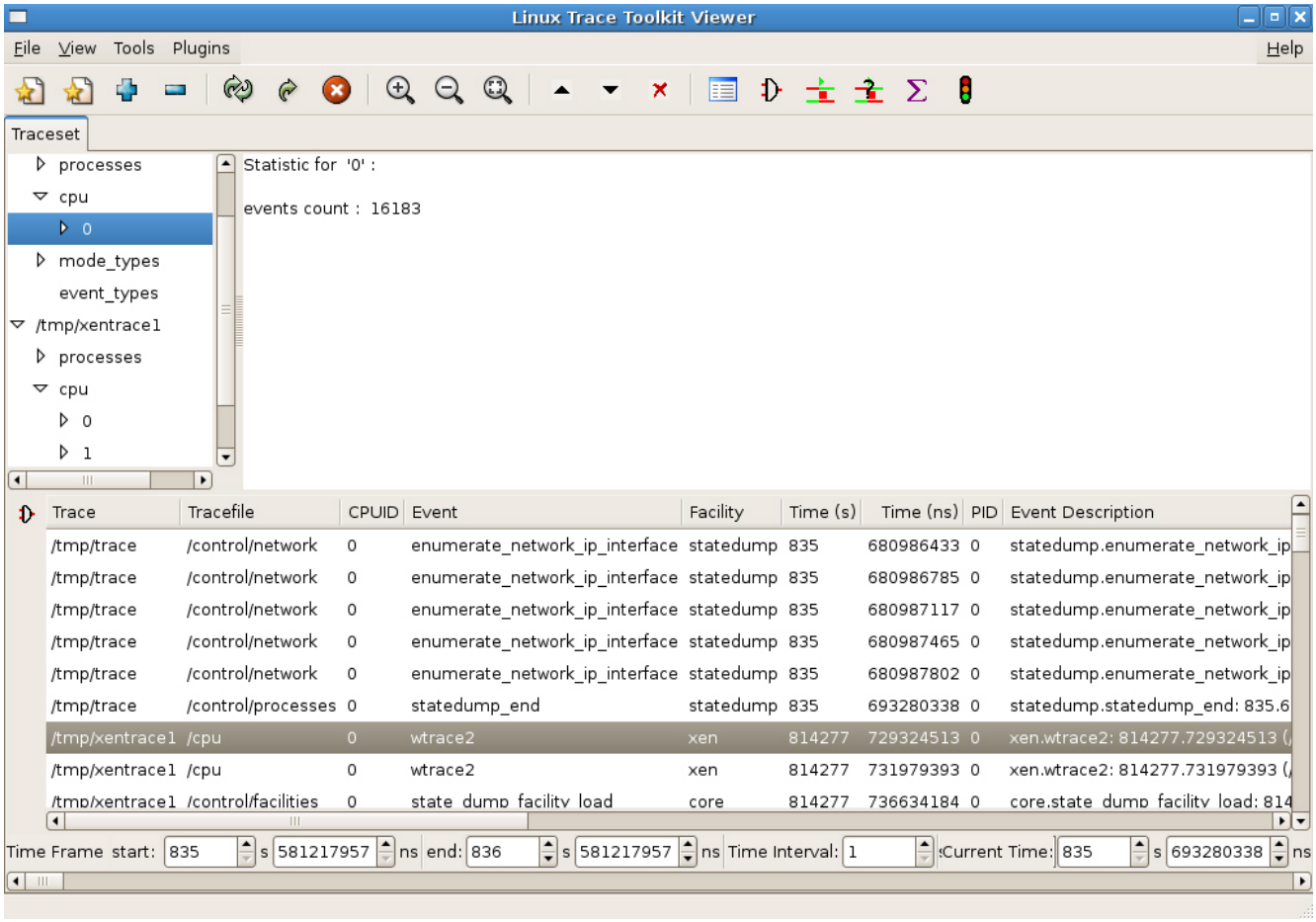

Fig. (3). Visualizing traces from domain0 and the hypervisor in the same window.

To keep the available memory identical for domain0, domainU and the physical machine, the same amount of memory is reserved for the unmodified kernel, the modified kernel of domain0, and the virtual machines, e.g. 900MB each, a subset of the 2GB physically installed on the computer.

Xen-unstable[2], was used as hypervisor and vanilla Linux version 2.6.19 was used for the physical machine. Modified versions of Xen compatible Linux 2.6.19 provided with Fedora [14] were used for all virtual machines. LTTng patches version 0.6.77 were back-ported to Linux 2.6.19.

Each test has been repeated several times to assess the variability of the measured times. The results presented in the tables below are a geometric mean of at least three repeats. For each test, the standard deviation of ten repeats of step 0, Linux native, is shown to give a better idea of the relative impact caused by LTTng. Results are presented as absolute numbers as well as the percentage of performance loss versus the real machine. Bar charts show graphically the relative performance in percentage of the systems incurring virtualization and tracing overhead. The horizontal axis

At each iteration (reboot step):
- Boot the selected operating system.
- If "stepnumber.txt" exists, read STEPNUMBER and execute the main test script.
  - Create a virtual machine with physical disk if STEPNUMBER is 8, 9, 10, 11.
  - Create a virtual machine with virtual disk if STEPNUMBER is 12, 13, 14, 15.
  - Start LTTng in flight recorder mode if STEPNUMBER is 2, 6, 10, 14.
  - Start trace and write it to disk if STEPNUMBER is 3, 7, 11, 15.
  - Start Xentrace if STEPNUMBER is 16.
  - Start LTTng-Xen if STEPNUMBER is 17.
- Run the appropriate test file (determined in definition file).
- Save results.
- Based on STEPNUMBER select the operating system to reboot into for the next iteration:
  - Linux with LTTng.
  - Linux on Xen Platform.
  - Linux with LTTng on Xen platform.
  - Linux original.
- Increment STEPNUMBER by one and rewrite into the file, "stepnumber.txt"
- If STEPNUMBER > last step (17) remove "stepnumber.txt"
- Reboot the system.

Fig. (4). The benchmark procedure in pseudo code.

[2] Change set 14206:3ac19fda0bc2 http://xenbits.xensource.com/xen-unstable.hg

reflects the four rows of the corresponding tables. Numbers 1 to 4 correspond to the real machine, Xen-domain0, Xen-domainU (real disk) and Xen-domainU (virtual disk), respectively.

5.1. Compiling

The first test consists in compiling the Linux kernel 2.6.15.4 with optimization level 2, a mostly CPU intensive task. It is presented in Fig. (**5**) and Table **1**, each bar in Fig. (**5**) corresponds to a cell of Table **1**. Surprisingly, the last column of Table **1** shows that tracing and recording the trace information to disk takes less time than tracing alone in a virtual machine with a virtual disk. To better understand this, a simple test was run on a virtual machine (domainU with real disk), with LTTng kernel, probes loaded, but tracing disabled, while a short script running in the background simulates recording a trace by simply appending periodically 1MB of data to a file. In this test, the performance is similarly improved (Table **2**). Therefore, the unexpected speedup is not the effect of tracing, but a border effect of the I/O scheduler and Xen.

The cost associated with tracing is less than 3%, even though a 500MB trace was generated in 372.89s for cache hot (1.34MB/s). This is a very minor disturbance considering

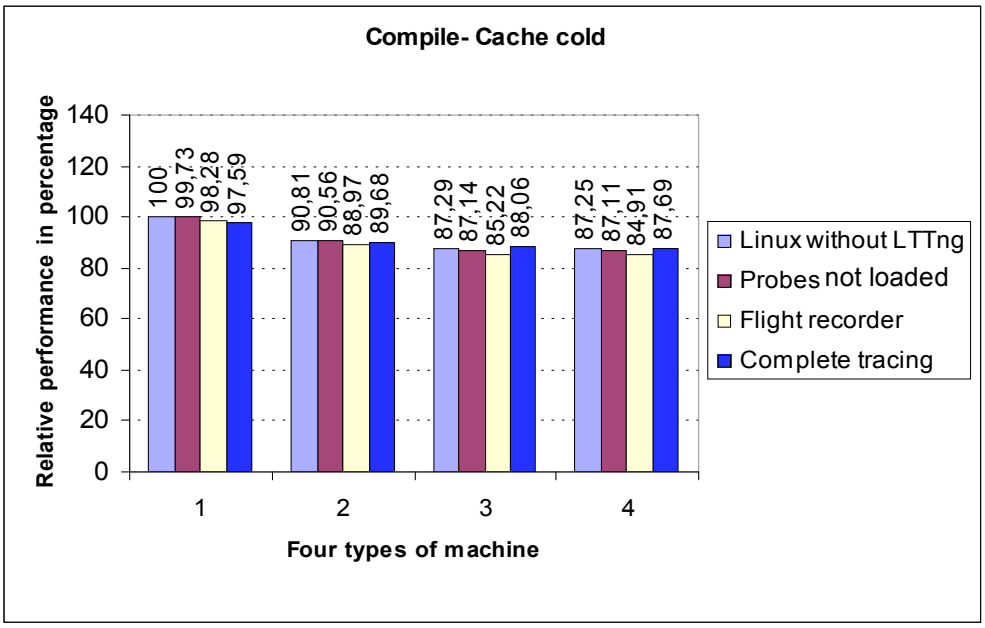

Fig. (5). Relative performance in percentage - Compile - Cache cold.

Table 1. Compilation - Tracing Domains - Cache Hot

Cache Hot (Minute)	LTTng is Not Compiled	LTTng Compiled, Probe Not Loaded	Tracing in Flight Mode	Trace is Active
Real machine (Original Linux)	6:04.93 (100%) *(σ =0.85sec)*	6:05.58 (99.82%)	6:11.48 (98.20%)	6:12.89 (97.82%)
Xen-dom0	6:38.96 (90.67%)	6:39.04 (90.65%)	6:45.64 (88.84%)	6:42.29 (89.76%)
Xen-domU (real disk)	6:54.10 (86.52%)	6:54.54 (86.41%)	7:02.41 (84.25%)	6:49.15 (87.88%)
Xen-domU (virtual disk)	6:53.00 (86.83%)	6:53.82 (86.60%)	7:01.52 (84.49%)	6:51.32 (87.29%)

Table 2. Simulating the Trace Effect – Compilation

Cache Cold (Minutes)	LTTng is Not Compiled	LTTng Compiled Probes Not Loaded	Tracing in Flight Mode	Trace is Active	Write into a File, Probes Loaded
Xen-domU	7:00.40	7:01.12	7:08.56	6:59.59	6:55.83

the accuracy and completeness of the information extracted. Having LTTng compiled in, but probes not loaded, causes very little impact.

In this test, tracing the hypervisor with the new proposed LTTng-Xen and with the existing Xentrace shows similar performance overhead, as shown at the end of Section 5. The Xentrace trace file occupies 17 MB while the LTTng-Xen file is smaller at 11MB. Xentrace keeps a fixed record size for all events. Events may have from 0 to 5 parameters. When an event has less than 5 parameters, Xentrace fills the record with 0. LTTng-Xen uses a more space efficient variable record size.

The Compile test is CPU intensive. The CPU virtualization by Xen, mapping virtual CPUs to physical CPUs, causes a measurable impact on the performance of 9%, which is three times as much as tracing (Compare the second column of the third row with the third column of the second row in Table **1**).

5.2. Archiving (Tar Create - Tar Extract)

Archiving uses the file system intensively. A large directory of 1.2 GB, containing the OpenOffice source code, is used as input for 'tar create' and a large tar file with the same content, is extracted. Fig. (**6**) and Table **3** correspond to tar create while Fig. (**7**) and Table **4** correspond to tar extract.

Because of the huge amount of data being written in each step, and especially since the archive is larger than the available memory, it is understandable that the second repeat

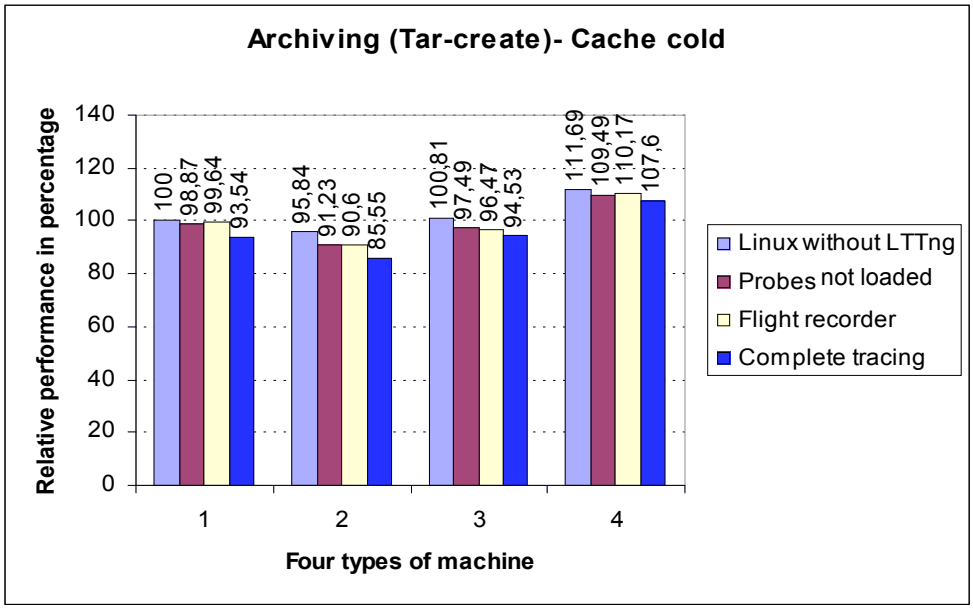

Fig. (6). Relative performance in percentage - Tar create - Cache cold.

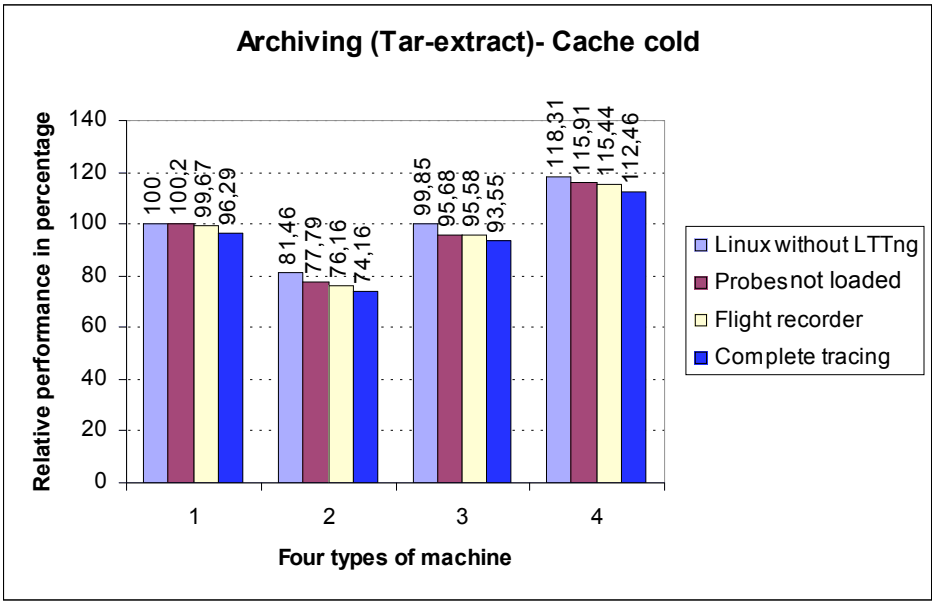

Fig. (7). Relative performance in percentage - Tar extract - Cache cold.

Table 3. Tar Create - Tracing Domains - Cache Hot

Cache Hot (Minute)	LTTng is Not Compiled	LTTng Compiled, Probe Not Loaded	Tracing in Flight Mode	Trace is Active
Real machine (Original Linux)	1:49.69 (100%) *(σ =1.56sec)*	1:50.16 (99.57%)	1:49.64 (100.05%)	1:55.15 (95.02%)
Xen-dom0	1:53.25 (96.75%)	1:56.39 (93.89%)	1:58.11 (92.32%)	2:03.85 (87.09%)
Xen-domU (real disk)	1:47.69 (101.82%)	1:51.37 (98.47%)	1:51.57 (98.29%)	1:55.89 (94.35%)
Xen-domU (virtual disk)	1:33.18 (115.05%)	1:36.98 (111.59%)	1:37.75 (110.89%)	1:42.89 (106.2%)

Table 4. Tar Extract – Tracing Domains - Cache Hot

Cache Hot (Minute)	LTTng is Not Compiled	LTTng Compiled, Probe Not Loaded	Tracing in Flight Mode	Trace is Active
Real machine (Original Linux)	1:37.07 (100%) *(σ =2.71sec)*	1:36.95 (100.12%)	1:36.94 (100.13%)	1:45.04 (91.79%)
Xen-dom0	1:57.99 (78.45%)	1:58.41 (78.01%)	2:00.47 (75.89%)	2:03.83 (72.43%)
Xen-domU (real disk)	1:36.58 (100.5%)	1:38.80 (98.22%)	1:37.38 (99.68%)	1:41.49 (95.45%)
Xen-domU (virtual disk)	1:37.99 (99.05%)	1:39.83 (97.16%)	1:42.57 (94.33%)	1:46.97 (89.8%)

sometimes takes longer than the first. Indeed, more free memory pages in the kernel are available to be used as write buffer the first time than the second one. The effect of having the LTTng event probes compiled in, and even the writing of event data in buffers in flight recorder mode, is barely measurable, being much less than the variance. The LTTng trace files occupy 290MB each for a native Linux system and for a Xen domain0, while it is about 170MB for domainU.

The trace file for the hypervisor layer is about 88MB with Xentrace and 63MB with LTTng-Xen. In this test, a large number of events occur in the hypervisor layer, causing a more significant impact and better showing the performance difference between LTTng-Xen and Xentrace at 1% and 6% overhead respectively.

It may be surprising to note that the tests with domainU show performance improvements. When a virtual machine needs to read from or write to the disk, its requests are handled by domain0. In our configuration, each of domain0 and domainU has its own 900MB of RAM, providing much more total space usable for various I/O buffering tasks.

For tar extract as well, neither the probes nor writing the events to the buffers (in flight recorder mode) has a significant impact on performance. Writing the events to the trace file causes a 4% overhead for cache cold while it is 8% for cache hot as the disk subsystem is already fully utilized by the test application. It is interesting to note that adding LTTng without loading the probes makes the test faster. However, the difference is minimal and much smaller than the standard deviation. The trace file is about 180MB on the real machine and domain0, and 150MB on domainU. In the hypervisor layer, the trace file is about 45MB with LTTng-Xen and 60 MB with Xentrace. The impact of tracing the hypervisor is small but LTTng-Xen remains more efficient than Xentrace.

5.3. Compression

Compression of an archived file (.tar) using bzip2 is a mostly CPU intensive application. For this test, the Linux 2.6.16 source code which occupies about 200 MB was chosen. Corresponding results are represented in Fig. (**8**) and Table **5**. The Trace file for domain0, as well as for the real machine, is about 80 MB, and 60 MB for virtual domains. The hypervisor layer trace file is 9.1 MB with LTTng-Xen and 15MB with Xentrace.

The size of the file to compress is smaller than available memory in all cases; thus, once it is read, it should remain available in buffer cache for the next repeat (cache hot). It is interesting to see that LTTng and tracing cause more impact on Linux original in cache cold rather than cache hot. A possible explanation is the memory usage caused by the LTTng buffers. When the system is running a cache hot test, some piece of data is already in the cache. LTTng also uses the cache for its buffers. Thus, when tracing, less space is

Table 5. Compression - Tracing Domains - Cache Hot

Cache Hot (Minute)	LTTng is Not Compiled	LTTng Compiled, Probe Not Loaded	Tracing in Flight Mode	Trace is Active
Real machine (Original Linux)	1:16.55 (100%) *(σ =0.30sec)*	1:18.55 (97.33%)	1:18.49 (97.46%)	1:19.02 (96.77%)
Xen-dom0	1:21.03 (94.15%)	1:21.30 (93.79%)	1:22.22 (92.59%)	1:22.35 (92.42%)
Xen-domU (real disk)	1:18.89 (96.94%)	1:19.51 (96.13%)	1:20.18 (95.26%)	1:20.28 (95.12%)
Xen-domU (virtual disk)	1:18.88 (96.96%)	1:19.24 (96.49%)	1:19.91 (95.61%)	1:20.09 (95.37%)

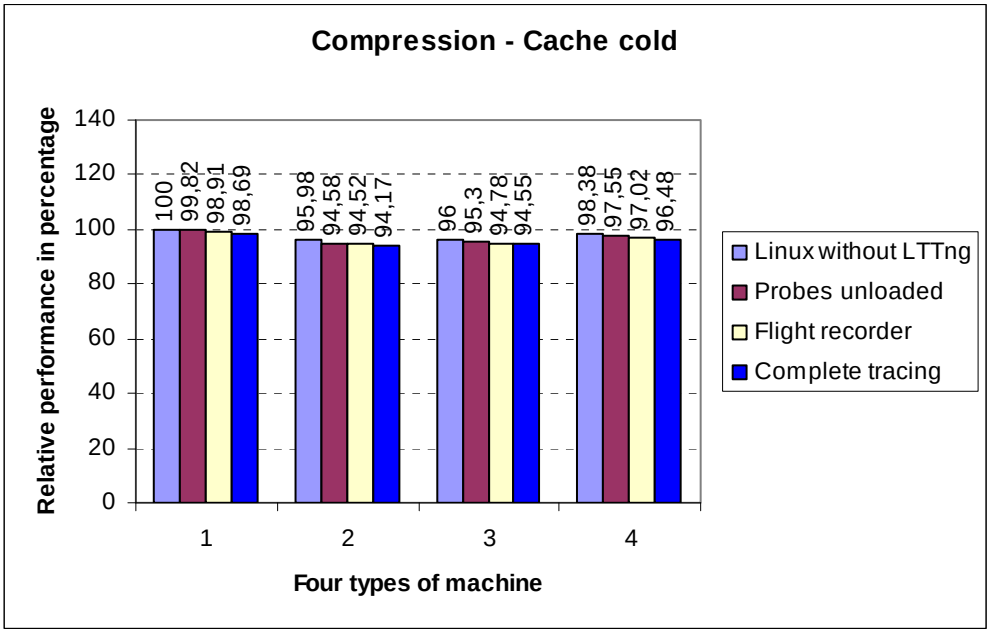

Fig. (8). Relative performance in percentage - Compression - Cache cold.

free in the cache and the impact of tracing in cache hot is more than for the case of cache cold.

5.4. Dbench

Dbench [15] simulates load patterns of file systems. For this test, all values are throughput in MB/sec and represented in Fig. (**9**) and Table **6**. The trace file is about 4.8 GB on physical machine and Xen-domain0 and 1.8 GB on domainU. It is also about 13 MB for tracing the hypervisor by Xentrace and 7.1 MB by LTTng-xen.

In this case, Xen does not cause a large impact. Here also, the results of domainUs are better than native Linux,

Table 6. Dbench - Tracing Domains - Cache Hot

Cache Hot (Minute)	LTTng is Not Compiled	LTTng Compiled, Probe Not Loaded	Tracing in Flight Mode	Trace is Active
Real machine (Original Linux)	216.70 (100%) *(σ =22.95)*	213.84 (98.68%)	199.60 (92.11%)	193.85 (89.46%)
Xen-dom0	211.61 (97.65%)	202.16 (93.29%)	191.50 (88.37%)	182.71 (84.31%)
Xen-domU (real disk)	228.32 (105.36%)	223.92 (103.34%)	226.07 (104.32%)	221.96 (102.42%)
Xen-domU (virtual disk)	231.7 (106.92%)	231.67 (106.91%)	227.06 (104.77%)	226.93 (104.73%)

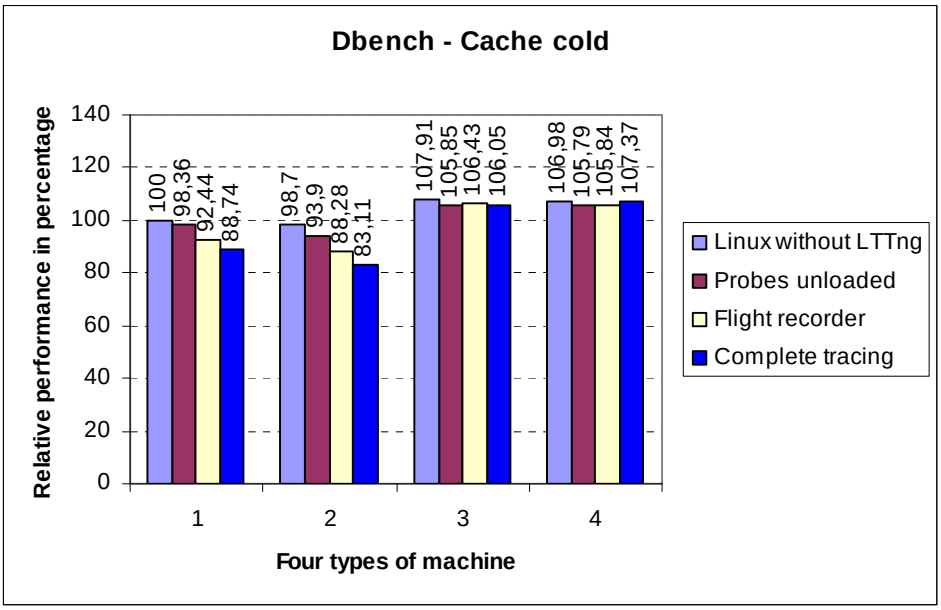

Fig. (9). Relative performance in percentage - Dbench - Cache cold.

because of the larger total amount of RAM contained in domain0 and domainU together. In the real machine as well as domain0, the effect of tracing is more than 10%. Indeed, Dbench is a very intense test and stresses strongly the disk and memory subsystems.

5.5. SPECviewperf9.1

SPECviewperf9.1 [16] is chosen as a graphical benchmark which measures OpenGL 3D graphic characteristics. Considering the limitations of graphics acceleration in virtual domains, this test is only executed on

Table 7. SPECviewperf9.1 - Tracing Domains

Cache Hot (Minute)	LTTng is Not Compiled	LTTng compiled, Probe Not Loaded	Tracing in Flight Mode	Trace is Active
Real machine (Original Linux)	1.071 (100%)	1.056 (98.60%)	1.031 (96.26%)	1.023 (95.55%)
Xen-dom0	0.987 (92.14%)	0.963 (89.95%)	0.934 (87.16%)	0.927 (86.58%)

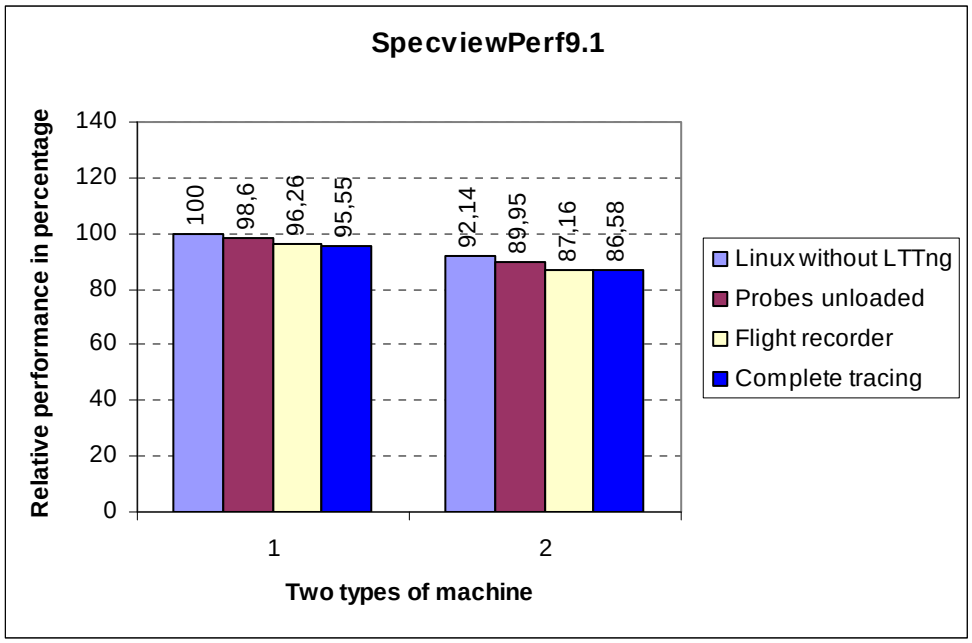

Fig. (10). Relative performance in percentage - SPECviewperf9.1 - Cache cold.

real machines and domain0. SPECviewperf9.1 calculates the result in frames per second and contains multiple view sets, among which we have chosen 'Maya'. SpecviewPerf9.1 repeats each test a few times. The file system is not used much in this test and thus, cache hot/cold is not considered in this case. Corresponding results are represented in Fig. (**10**) and Table **7**.

LTTng causes less than 2% loss in performance when probes are not loaded and less than 5% while writing the trace, a reasonable impact. Since domain0 normally has direct access to the hardware, the 8% graphics performance loss seems significant. It should be noted, however, that Xen has mostly been optimized for non graphical server applications.

5.6. LMbench

LMbench [17] consists in a set of specialized tests, each measuring a specific characteristic of the system, either in terms of bandwidth (operations per second) or in terms of latency (delay between sending the request and receiving the answer). It covers create/delete a filesystem, signal handling, create/delete a process, cached read/write, memory operations like copy/read/write and others. LMbench categorizes each test into a group such as Hardware or OS (Operating System). Our focus here is on the OS category.

Some of the metrics observed by LMbench (Process category) show a high performance loss with both LTTng and Xen. Some operations like fork, exec and sh require a lot of page table updates to be verified by Xen, thus causing a large impact on performance [3]. On the other hand, several system calls are simple and fast, making the event recording overhead significant. In particular, the Select system call generates several events per call, making the tracing overhead even worse. Results presented here in Tables **8**, **9**, **10** and **11** are some of the metrics chosen from the Local communications category. Numbers show the bandwidths in MB/Sec where bigger values are better. Here also, in some I/O applications recording a trace to the disk increases the performance.

The comparison between the results of tracing the hypervisor by LTTng-Xen and Xentrace for LMbench is given in Table **12**. More details and information are found in [18]. Fig. (**11**) and Table **13** show a comparison between the impact imposed by LTTng-Xen and that of Xentrace. On the horizontal axis of Fig. (**11**), numbers 1 to 6 represent the results of compile, tar-create, tar-extract, compression, Dbench, and specview9.1 respectively.

6. VIRTUALIZATION AND LOAD DISTRIBUTION

Running different instances of an application in different virtual machines provides stronger security and possibly fair resource allocation through isolation. The question, however, is the performance overhead of running several virtual machines. Two different cases were tested:

- Domain0 idle, four domainUs, each of them running a single task.

Table 8. LMbench (Local Communication) - Real Machine

Real Machine (MB/sec)	Pipe	File Reread	Mmap Reread	Mem Read	Mem Write
LTT is not compiled	1668.7 (100%) *(σ = 4.21)*	1896 (100%) *(σ = 8.26)*	4284.3 (100%) *(σ = 3.4)*	4288 (100%) *(σ =2.17)*	1952 (100%) *(σ =1.32)*
LTT compiled, probes not loaded	1643 (98.46%)	1882.8 (99.30%)	4282 (99.94%)	4280 (99.81%)	1950 (99.90%)
flight tracing	1568 (93.96%)	1846.9 (97.41%)	4277 (99.83%)	4282 (99.86%)	1950 (99.90%)
tracing	1524 (91.33%)	1867.7 (98.50%)	4290.7 (100.15%)	4311 (100.53%)	1954 (100.10%)

Table 9. LMbench (Local Communication) – Domain0

Domain0 (MB/sec)	pipe	File Reread	Mmap Reread	Mem Read	Mem Write
LTT is not compiled	1616 (96.84%)	4266 (99.57%)	4266 (99.57%)	4269 (99.56%)	1837 (94.10%)
LTT compiled, probes not loaded	1618 (96.96%)	4260.9 (99.45%)	4260.9 (99.45%)	4274 (99.67%)	1836 (94.06%)
flight tracing	1530 (91.69%)	4259.2 (99.41%)	4259.2 (99.41%)	4258 (99.30%)	1835 (94.00%)
tracing	1027 (61.54%)	4281.7 (99.94%)	4281.7 (99.94%)	4301 (100.30%)	1839 (94.21%)

Table 10. LMbench (Local Communication) – DomainU with Real Disk

DomainU-Real Disk (MB/sec)	Pipe	File Reread	Mmap Reread	Mem Read	Mem Write
LTT is not compiled	1611 (96.54%)	835.9 (44.09%)	4253.6 (99.28%)	4252 (99.16%)	1932 (98.97%)
LTT compiled, probes not loaded	1613 (96.66%)	834.6 (44.02%)	4247.5 (99.14%)	4250 (99.11%)	1929.6 (98.85%)
flight tracing	1608 (96.36%)	837.9 (44.19%)	4249 (99.17%)	4252 (99.16%)	1932 (98.97%)
tracing	1600 (95.88%)	841.3 (44.37%)	4246.2 (99.11%)	4246 (99.02%)	1929 (98.82%)

Table 11. LMbench (Local Communication) – DomainU with Virtual Disk

DomainU-Virtual Disk (MB/sec)	Pipe	File Reread	Mmap Reread	Mem Read	Mem Write
LTT is not compiled	1605.6 (96.22%)	850 (44.83%)	4243.8 (99.05%)	4246 (99.02%)	1931 (98.92%)
LTT compiled, probes not loaded	1612 (96.60%)	848 (44.72%)	4246.2 (99.11%)	4243 (98.95%)	1932 (98.97%)
flight tracing	1612 (96.60%)	878.3 (46.32%)	4249.7 (99.19%)	4231 (98.67%)	1932 (98.97%)
Tracing	1610 (96.48%)	880.9 (46.46%)	4244.3 (99.06%)	4247 (99.04%)	1930 (98.87%)

Table 12. LMbench (Local Communication) – Hypervisor

Hypervisor (MB/sec)	Pipe	File Reread	Mmap Reread	Mem Read	Mem Write
Domain0	1616 (96.84%)	1261 (66.50%)	4266 (99.57%)	4269 (99.56%)	1837 (94.11%)
LTTng-Xen	1620 (97.08%)	1236 (65.19%)	4265.7 (99.56%)	4265 (99.46%)	1837 (94.11%)
Xentrace	1619 (97.02%)	1280.8 (67.55%)	4257 (99.36%)	4265 (99.46%)	1833 (93.90%)

- Domain0 idle, one domainU (having 1 VCPU) running four tasks in parallel.

Four tasks are being executed; their sources are located on four different partitions close to each other. However, the geometry of the disk and the heads displacements cause a slight impact on the result. The creation time of the virtual machines is not taken into account in the measurements.

Two applications, compilation and archiving, are tested. The first column of Table **14** contains the results of creating four virtual machines concurrently and running a compilation task on each of them. The second column shows the results of creating a single virtual machine and running the same four tasks in parallel. In both tests, all tasks start at the same time. CPU usage is 24% or 25% for all, whether one or four tasks per machine. In order to simplify the comparison, all of four virtual machines are mono-processor (VCPU=1) and all of VCPUs are mapped on a unique CPU. Thus, this improvement is not because of having more available resources, rather it is solely the effect of Xen. The time spent to run four tasks in parallel on a single virtual machine is slightly more than running them individually, probably because of a larger scheduling granularity between virtual machines.

The second test is an I/O intensive task, tar-create, presented in Table **15**. Available RAM and location of the source files are the same as 'compilation'. Here again, the performance is better when using four virtual machines than four tasks in a single virtual machine.

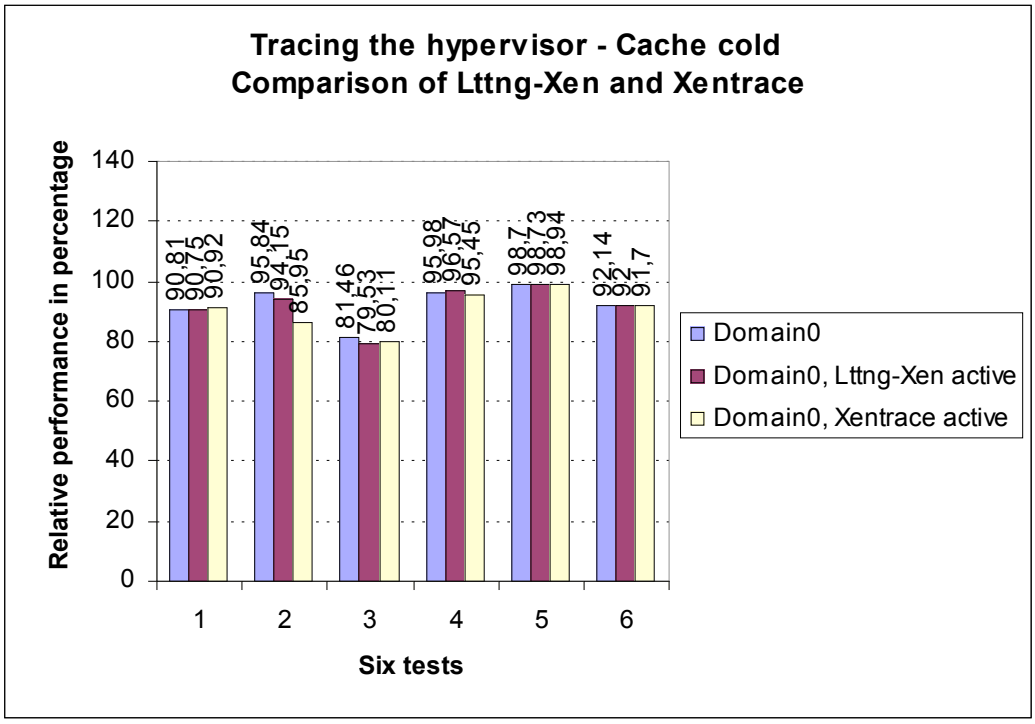

Fig. (11). Comparison of LTTng-Xen and Xentrace.

Table 13. Comparison of LTTng-Xen and Xentrace

Cache Hot	LTTng is Not Compiled Xentrace is Inactive	LTTng-Xen is Active	Xentrace is Active
Compilation (minute)	6:38.96 (90.67%)	6:38.25 (90.87%)	6:37.51 (91.07%)
Tar-create (minute)	1:53.25 (96.75%)	1:54.19 (95.90%)	2:00.43 (90.21%)
Tar-extract (minute)	1:57.99 (78.45%)	2:01.16 (75.18%)	2:02.57 (73.73%)
Compression (minute)	1:21.03 (94.15%)	1:21.13 (94.01%)	1:20.83 (94.41%)
Dbench (MB/sec)	211.61 (97.65%)	207.41 (95.71%)	213.73 (98.63%)
SPECviewperf9.1 (Frame/sec)	0.987 (92.14%)	0.985 (92.00%)	0.982 (91.70)

Table 14. Compiling, Four Tasks on Four Partitions - Virtual Machine

Compile (Minute)	Four Tasks, One Per Machine	Four Tasks on One Virtual Machine Having 1 VCPU
task 1	26:34.41	27:58.78
task 2	26:39.89	28:03.92
task 3	26:27.87	27:19.16
task 4	26:37.71	28:02.63

Table 15. Archiving, Four Tasks on Four Partitions - Virtual Machine

Tar-Create (Minute)	Each Task on a Virtual Machine	Four Tasks on One Virtual Machine having 1 VCPU
task 1	10:18.05	12:13.85
task 2	10:55.29	12:48.18
task 3	11:04.55	12:44.15
task 4	11:04.48	12:51.17

CONCLUSION

The results obtained demonstrate that for typical server applications, virtual machines bring a performance overhead lower than 5% in most of the cases. In some I/O tests, like tar archiving, their impact may exceed 10% in domain0. However, because of different I/O scheduling and of the combined domainU/domain0 memory available for buffering, in some cases the same I/O tests performed even better in domainU virtual machines than on a physical machine.

The performance of systems with intense interaction between the applications and the operating system is particularly difficult to analyze. Tracing is often the most detailed and accurate source of information for this purpose. The difficulty is even more acute with virtualization where several virtual machines interact with an hypervisor in order to share the resources of the physical machine. The Linux Trace Toolkit was ported to the domain0 and domainU virtual machines and extended to trace the Xen hypervisor (LTTng-Xen). It was then possible to obtain a complete picture of all the important events happening on a Xen system running several Linux virtual machines. This extension replaces the current simpler tracing system provided with Xen, Xentrace, and provides the complete information with a single tracing system, using a common time base.

The performance overhead associated with tracing, with and without virtual machines, was measured using the new proposed test program. The tracing overhead remains almost negligible at between 2% and 5%, with little or no impact on scheduling or real-time response, because of the atomic operations used in LTTng. The performance of Xentrace and LTTng-Xen was specifically compared. LTTng uses a variable size event format and thus produces more compact traces. Furthermore, the associated overhead for LTTng-Xen was smaller than for Xentrace in most cases.

In this document, we have studied virtual machines based on the Xen hypervisor as a solution for consolidating and simplifying the management of server applications, and measured the corresponding performance overhead. A new automated performance measurement procedure was proposed and used to evaluate various characteristics of system performance in different contexts. Our procedure is easily extensible to cover more characteristics and system configuration alternatives. It would be interesting in the future to complement the system with a graphical reporting system, converting the raw performance data into various graphs and statistics.

LTTng-Xen currently reuses the events defined for Xentrace. The events could be redefined specifically for LTTng-Xen to benefit from more concise event typing, and extended to trace more information such as virtual CPUs.

ACKNOWLEDGMENTS

The financial support of the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

REFERENCE

[1] A. Padegs, "System/360 and beyond," *IBM J. Res. Dev.*, vol. 25 pp. 377-390, September 1981.

[2] SWSOFT, "Top ten considerations for choosing a server virtualization technology," 2006.

[3] 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 *SOSP'03: Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Lake George, NY, United States, pp. 164-177, 2003.

[4] I. Pratt, F. Keir, S. Hand, C. Limpach, A. Warfield, D. Magenheimer, J. Nakajima, and A. Mallick, "Xen 3.0 and the art of virtualization," in *Ottawa Linux Symposium*, Ottawa, Canada, pp. 65-77, 2005.

[5] M. Bligh, M. Desnoyers, and R. Schultz, "Linux kernel debugging on Google-sized clusters," In: *Ottawa Linux Symposium*, Ottawa, Canada, pp. 29-40, 2007.

[6] R. W. Wisniewski, R. Azimi, M. Desnoyers, M. M. Michael, J. Moreira, D. Shiloach, and L. Soares, "Experiences understanding performance in a commercial scale-out environment," In: *13th International Euro-Par Conference on Parallel Processing, Euro-Par*, Rennes, France, pp. 139-149, 2007.

[7] V. Prasad, W. Cohen, F. Eigler, M. Hunt, J. Keniston, and B. Chen, "Locating system problems using dynamic instrumentation," In: *Ottawa Linux Symposium*, Ottawa, Canada, pp. 49-64, 2005.

[8] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal, "Dynamic instrumentation of production systems," In: *Proceedings of the General Track 2004 USENIX Annual Technical Conference*, Boston, MA, 2004.

[9] K. Yaghmour, and M. R. Dagenais, "Measuring and characterizing system behavior using kernel-level event logging," In: *Proceedings of the 2000 USENIX Annual Technical Conference*, San Diego, CA, pp. 13-26, 2000.

[10] M. Desnoyers and M. R. Dagenais, "The LTTng tracer: A low impact performance and behavior monitor of GNU/Linux," In: *Ottawa Linux Symposium*, Ottawa, Canada, pp. 209-223, 2006.

[11] "Linux Kernel Mailing List (LKML) ", 2006. Available: http://lkml.org /lkml/2006/9/ 15/148.

[12] P. Heidari, M. Desnoyers, and M. dagenais, "Performance analysis of virtual machines through tracing," In: *CCECE 2008*, Niagara Falls, Canada, pp. 261-266, 2008.

[13] M. R. Dagenais, "Disks, partitions,volumes and RAID performance with the Linux operating system," 2005, [Online] available http://arxiv.org /ftp/cs/papers/0508/0508063.pdf , [Accessed: May 31, 2007].

[14] "Fedora project homepage:" Available: http://download.fedora. redhat. com/pub/fedora/linux/core/updates/5/SRPMS, [Accessed: February 1st, 2007].

[15] A. Tridgella., "Dbench Readme" revision 2007, available at: http://samba.org/ftp/tridge/dbench/README, [Accessed: May 31, 2007].

[16] "Standard performance evaluation corporation" Available: www.spec.org, [Accessed: May 31, 2007].

[17] L. McVoy, and C. Staelin, "LMbench - Tools for performance analysis."Available: http://lmbench.sourceforge.net/, [Accessed: May 31, 2007].

[18] P. Heidari, *"Hypervisors and virtual systems tracing for performance analysis,"* Masters' Thesis, École Polytechnique de Montreal, Montreal, 2007.

Received: March 24, 2010 Revised: September 15, 2010 Accepted: December 1, 2010