Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor

Ludmila Cherkasova, Rob Gardner
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2005-62
March 21, 2005*

E-mail: lucy.cherkasova@hp.com, rob.gardner@hp.com

virtual machine monitor, device drivers, I/O processing, web server and disk intensive workloads, monitoring framework, measurements, performance evaluation

Virtual Machine Monitors (VMMs) are gaining popularity in enterprise environments as a software-based solution for building shared hardware infrastructures via virtualization. In this work, using the Xen VMM, we present a light weight monitoring system for measuring the CPU usage of different virtual machines including the CPU overhead in the device driver domain caused by I/O processing on behalf of a particular virtual machine. Our performance study attempts to quantify and analyze this overhead for a set of I/O intensive workloads.

* Internal Accession Date Only
To be published in the Usenix '05 Annual Technical Conference, 10-15 April 2005, Anaheim, CA, USA
Approved for External Publication

Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor*

Ludmila Cherkasova
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94303, USA
lucy.cherkasova@hp.com

Rob Gardner
Hewlett-Packard Laboratories
3404 E Harmony Rd., Fort Collins, CO 80528, USA
rob.gardner@hp.com

Abstract. *Virtual Machine Monitors (VMMs) are gaining popularity in enterprise environments as a software-based solution for building shared hardware infrastructures via virtualization. In this work, using the Xen VMM, we present a light weight monitoring system for measuring the CPU usage of different virtual machines including the CPU overhead in the device driver domain caused by I/O processing on behalf of a particular virtual machine. Our performance study attempts to quantify and analyze this overhead for a set of I/O intensive workloads.*

1 Introduction

The current trend toward virtualized computing resources and outsourced service delivery has caused interest to surge in *Virtual Machine Monitors (VMMs)* that enable diverse applications to run in isolated environments on a shared hardware platform. The Xen virtual machine monitor [1] allows multiple operating systems to execute concurrently on commodity x86 hardware. The recent HP Labs SoftUDC project [3] is using Xen to create isolated virtual clusters out of existing machines in a data center that may be shared across different administrative units in an enterprise. Managing this virtual IT infrastructure and adapting to changing business needs is a challenging task. In SoftUDC, virtual machines (VMs) can be migrated from one physical node to another when current physical node capacity is insufficient, or for improving the overall performance of the underlying infrastructure.

To support these management functions, we need an accurate monitoring infrastructure reporting resource usage of different VMs. The traditional monitoring system typically reports the amount of CPU allocated by the scheduler for execution of a particular VM over time. However, this method might not reveal the "true" usage of the CPU by different VMs. The reason is that virtualization of I/O devices results in an I/O model where the data transfer process involves additional system components, e.g. hypervisor and/or device driver domains. Hence, the CPU usage when the hypervisor or device driver domain handles the I/O data on behalf of the particular VM needs to be charged to the corresponding VM.

In this work, we present a lightweight, non-intrusive monitoring framework for measuring the CPU overhead in VMM related layers during I/O processing and a method for charging this overhead to VMs causing the I/O traffic. Our performance study presents measurements of the CPU overhead in the device driver domain during I/O processing and attempts to quantify and analyze the nature of this overhead.

2 Xen

Xen [1, 2] is an x86 virtual machine monitor based on a virtualization technique called *paravirtualization* [8, 1], which has been introduced to avoid the drawbacks of full virtualization by presenting a virtual machine abstraction that is similar but not identical to the underlying hardware. Xen does not require changes to the application binary interface (ABI), and hence no modifications are required to guest applications. For full details on the Xen architecture and features, we refer readers to papers [1, 2]. Here, we only touch on some implementation details of Xen that are important for our monitoring framework and performance study.

In the initial design [1], Xen itself contained device driver code and provided safe shared virtual device access. The support of a sufficiently wide variety of devices is a tremendous development effort for every OS project. In a later paper [2], the Xen team proposed a new architecture used in the latest release of Xen which allows unmodified device drivers to be hosted and executed in isolated "driver domains" which, in essence, are driver-specific virtual machines.

There is an initial domain, called *Domain0*, that is created at boot time and which is permitted to use the control interface. The control interface provides the ability to create and terminate other domains, control the CPU scheduling parameters and resource allocation policies, etc. *Domain0* also may host unmodified Linux device drivers and play the role of a driver domain. In our experimental setup, described in Sec-

*Accepted to 2005 Usenix Annual Technical Conference.

tion 4, we use *Domain0* as a driver domain. Devices can be shared among guest operating systems. To make this sharing work, the privileged guest hosting the device driver (e.g. *Domain0*) and the unprivileged guest domain that wishes to access the device are connected together through virtual device interfaces using device channels [2]. Xen exposes a set of clean and simple device abstractions. I/O data is transferred to and from each domain via Xen, using shared-memory, asynchronous buffer descriptor rings. In order to avoid the overhead of copying I/O data to/from the guest virtual machine, Xen implements the "page-flipping" technique, where the memory page containing the I/O data in the driver domain is exchanged with an unused page provided by the guest OS. Our monitoring framework actively exploits this feature to observe I/O communications between the guest domains and the driver domains.

3 Monitoring Framework

To implement a monitoring system that accounts for CPU usage by different guest VMs, we instrumented activity in the hypervisor CPU scheduler.

Let $Dom_0, Dom_1, ..., Dom_k$ be virtual machines that share the host node, where Dom_0 is a privileged management domain (*Domain0*) that hosts the device drivers. Let Dom_{idle} denote a special idle domain that "executes" on the CPU when there are no other runnable domains (i.e. there is no virtual machine that is not-blocked and not-idle). Dom_{idle} is the analog to the "idle-loop" executed by an OS when there are no other runnable processes.

At any point of time, guest domain Dom_i can be in one of the following three states:

- *execution state*: domain Dom_i is currently using CPU;
- *runnable state*: domain Dom_i is not currently using CPU but is on the run queue and waiting to be scheduled for execution on the CPU;
- *blocked state*: domain Dom_i is blocked and is not on the run queue (once unblocked it is put back on the queue).

For each domain Dom_i, we collect a sequence of data describing the timing of domain state changes. Using this data, it is relatively straightforward to compute the share of CPU which was allocated to Dom_i over time.

As was mentioned in Section 2, in order to avoid the overhead of copying I/O data to/from the guest virtual machine Xen implements the "page-flipping" technique, where the memory page containing the I/O data is exchanged with an unused page provided by the guest OS. Thus, in order to account for different I/O related activities in Dom_0 (that "hosts" the unmodified device drivers), we observe the memory page exchanges between Dom_0 and Dom_i. We measure the number N_i^{mp} of memory page exchanges performed over time interval T_i when Dom_0 is in *execution state*. We derive the CPU cost (CPU time processing) of these memory page exchanges as $Cost_i^{mp} = T_i/N_i^{mp}$. After that, if there are $N_i^{Dom_i}$ memory page exchanges between Dom_0 and virtual machine Dom_i then Dom_i is "charged" for $N_i^{Dom_i} \times Cost_i^{mp}$ of CPU time processing of *Domain0*. In this way, we can partition the CPU time used by *Domain0* for processing the I/O related activities of different *VM*s sharing the same device driver, and "charge" the corresponding virtual machine that caused these I/O activities. Within the monitoring system, we use a time interval of 100 ms to aggregate overall CPU usage across different virtual machines.

4 Performance Study

We performed a few groups of experiments that exercise network and disk I/O traffic in order to evaluate the CPU usage caused by this traffic in *Domain0*.

All the experiments were performed on an HP x4000 Workstation with a 1.7 GHz Intel Xeon processor, 2 GB RAM, Intel e100 PRO/100 network interface, and Maxtor 40GB 7200 RPM IDE disk. For these measurements, we used the XenoLinux port based on Linux 2.6.8.1 and Xen 2.0.

The first group of experiments relates to web server performance. Among the industry standard benchmarks that are used to evaluate web server performance are the SPECweb'96 [6] and SPECweb'99 [7] benchmarks. The web server performance is measured as a maximum achievable number of connections per second supported by a server when retrieving files of various sizes. Realistic web server workloads may vary significantly in both their file mix and file access pattern. The authors of an earlier study [5] established the web server performance envelope: they showed that under a workload with a short file mix the web server performance is CPU bounded, while under a workload with a long file mix the web server performance is network bounded.

To perform a sensitivity study of the CPU overhead in *Domain0* caused by different web traffic, we use Apache HTTP server version 2.0.40 running in the guest domain, and the *httperf* tool [4] for sending the client requests. The *httperf* tool provides a flexible facility for generating various HTTP workloads and for measuring server performance. In order to measure the request throughput of a web server, we invoke *httperf* on the client machine, which sends requests to the server at a fixed rate and measures the rate at which replies arrive. We run the tests with monotonically increasing request rates, until we see that the reply rate levels off and the server becomes saturated, i.e., it is operating at its full capacity. In our experiments, the http client machine and web server are connected by a 100 Mbit/s network.

We created a set of five simple web server workloads, each retrieving a fixed size file: 1 KB, 10 KB, 30 KB, 50 KB, and 70 KB. Our goal is to evaluate the CPU overhead in *Domain0* caused by these workloads. Figure 1 summarizes the results of our experiments.

Figures 1 a), b) show the overall web server performance under the studied workloads. To present all the workloads on the same scale, we show the applied load expressed as a percentage of maximum achieved throughput. For example, the maximum throughput achieved under a workload with a 1 KB file size is 900 req/s. Thus the point on the graph with X axis of 100% reveals 900 req/s throughput shown on the Y axis. Similarly, the maximum throughput achieved under

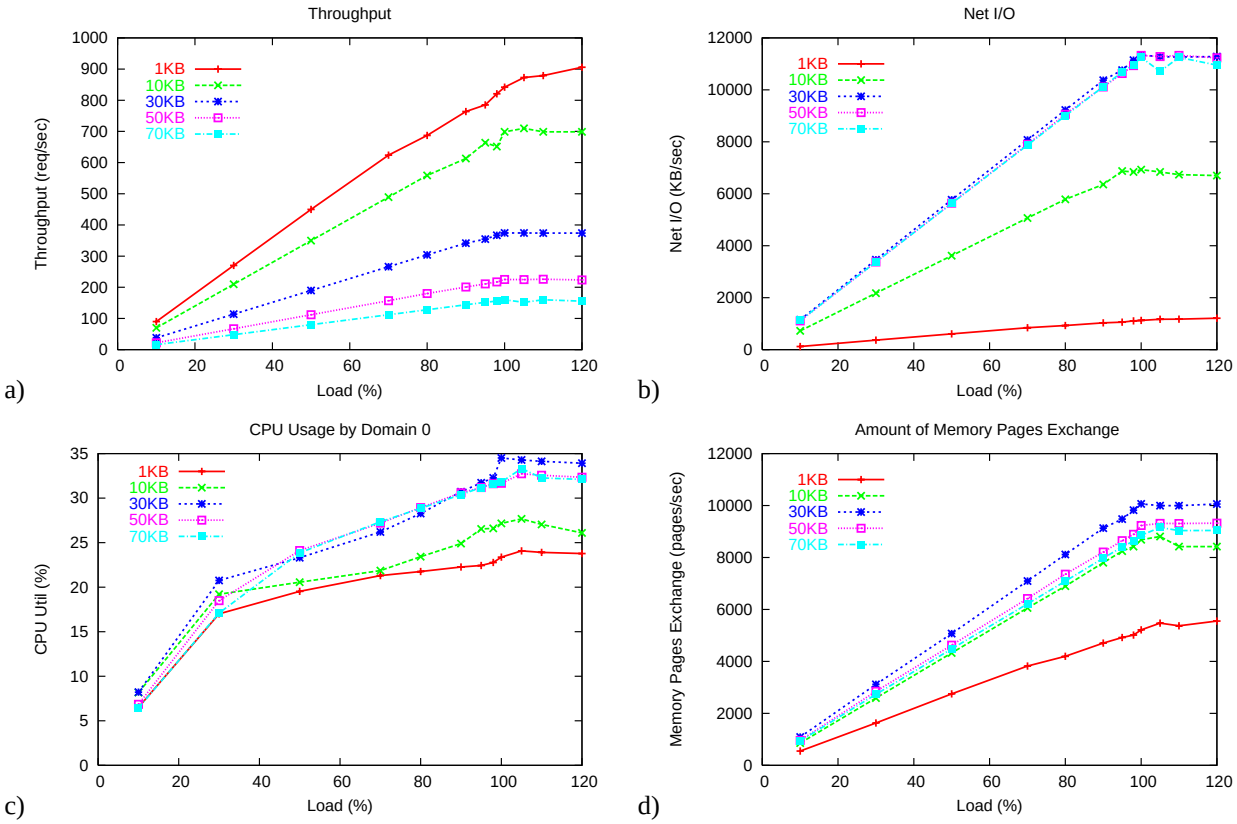

Figure 1: Summary of performance experiments with web server running in a single guest domain.

a workload with 70 KB file size is 160 req/s, and this point corresponds to 100% of applied load. Figure 1 b) presents the amount of performed network I/O in KB/s as reported by the *httperf* tool. These measurements combine the HTTP requests (80 bytes long) and 3the HTTP responses (that include 278 bytes HTTP headers and the corresponding file as content). Figure 1 b) reveals that web server throughput is network bounded for workloads of 30-70 KB files due to network bandwidth being limited to 100Mb/s (12.5 MB/s). Another interesting feature of these workloads is apparent from Figure 1 b): the amount of transferred network I/O is practically the same for workloads of 30-70 KB files (30KB$\times$380 req/s $\approx$ 50KB$\times$ 225 req/s $\approx$ 70KB$\times$160 req/s).

Figure 1 c) shows the measured CPU usage by *Domain0* for each of the corresponding workloads. The CPU usage by *Domain0* increases with a higher load, reaching 24% for the workload of 1 KB files and peaking at 33-34% for workloads of 30-70 KB files. Measurements presented in Figure 1 c) answer one of our questions about the amount of CPU usage in *Domain0* that is caused by device driver processing for web server related workloads. The measured CPU usage presents a significant overhead, and thus should be charged to the virtual machine causing this overhead.

Our monitoring tool measures the number of memory page exchanges performed per second over time. Figure 1 d) presents the rates of memory page exchanges between *Domain0* and the corresponding guest domain. At first glance, these numbers look surprising, but under more careful analysis, they all make sense. While the memory pages are 4 KB, a single memory page corresponds to a network packet transfer whether it is a *SYN*, or *SYN-ACK*, or a packet with the HTTP request. Thus network related activities for an HTTP request/response pair for a 1 KB file require at least 5 TCP/IP packets to be transferred between the client and a web server. Thus, processing 1000 requests for a 1 KB file translates to $\approx$ 5000 TCP/IP packets and the corresponding rates of memory page exchanges.

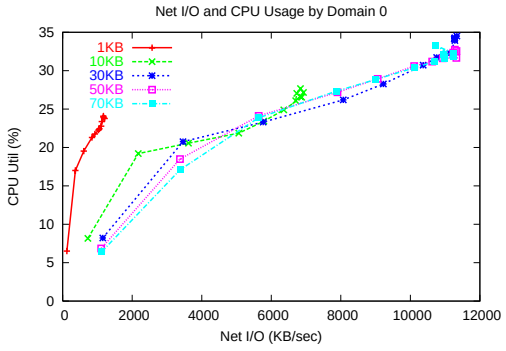

Figure 2: CPU usage by *Domain0* versus the amount of network I/O traffic transferred to/from a web server.

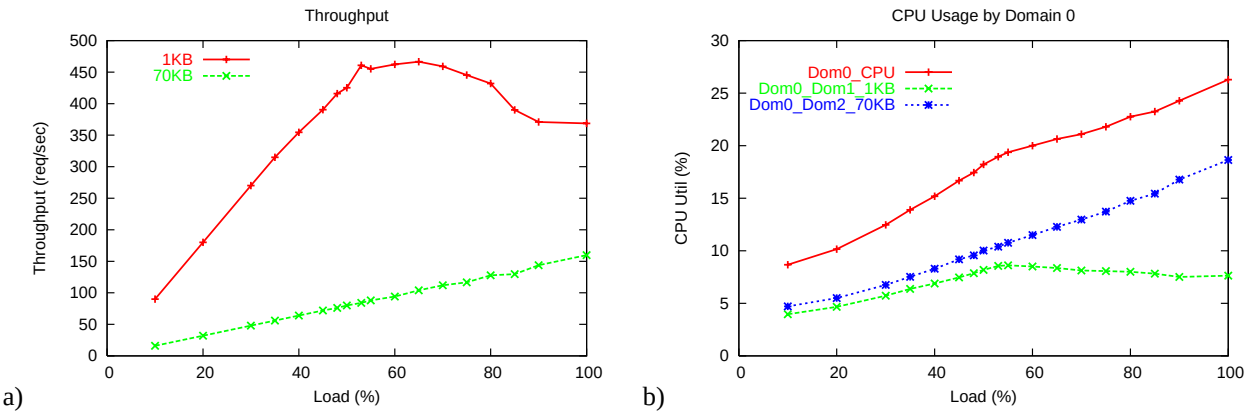

Figure 3: Summary of performance experiments with web servers running in two different guest domains.

Figure 2 presents the CPU usage by *Domain0* versus the amount of network I/O traffic transferred to/from a web server during the the corresponding workload. Often, the expectations are that the CPU processing overhead can be predicted from the number of transferred bytes. In the case of high volume HTTP traffic requesting small files, the small size transfers are "counter-balanced" by the high number of interrupts corresponding to the processing of the web requests. While the amount of transferred data is relatively small for the 1 KB file workload, it results in high CPU processing overhead in *Domain0* due to the vastly higher number of requests (10-20 times) for the corresponding workload as shown in Fig. 2.

To complete our web server case study, we run Xen with two guest domains configured with equal resource allocations, where each guest domain runs an Apache web server. Using the *httperf* tool, we designed a new experiment where requests that are sent to a web server in *Domain1* retrieve 1 KB files, and the requests sent to a web server in *Domain2* retrieve 70 KB files. In these experiments, we use the same request rates as in a single guest domain case. Our goal is to evaluate the CPU overhead in *Domain0* caused by these workloads, as well as present the parts of this overhead attributed to *Domain1* and *Domain2*.

Figure 3 a) presents throughput achieved for both web servers under the applied workloads. It shows that a web server running in *Domain1* and serving 1 KB file workload is able to achieve 50% of the throughput achieved by a web server running in a single guest domain (see Figure 1 a). Interestingly, a web server running in *Domain2* and serving 70 KB file workload is able to achieve almost 100% of its throughput compared to the single guest domain case. This is because the performance of a web server handling a 70 KB file workload is network bounded, not CPU bounded. In the designed experiment, it can use most of the available network bandwidth, since the "competing" 1 KB file workload is CPU bounded and has network bandwidth requirements that are 70 times lower.

Figure 3 b) shows the measured CPU usage by *Domain0* and the portions attributed to *Domain1* and *Domain2*. For example, under 100% of applied load, the CPU usage by *Domain0* is 26.5%, where *Domain1* is responsible for 7.5% and *Domain2* accounts for the remaining 19% of it. Thus, it is important to capture the additional CPU overhead caused by the I/O traffic in *Domain0*, and accurately charge it to the domain causing this traffic.

The second group of experiments targets the disk I/O traffic in order to evaluate the CPU usage in *Domain0* caused by this traffic. We use the *dd* command to read 500, 1000, 5000, and 10000 blocks of size 1024 KB from the "raw" disk. Table 1 summarizes the measurements collected during these experiments. First of all, the transfer time is directly proportional to the amount of transferred data. The achieved disk bandwidth, the *Domain0* CPU usage, and the rates of memory page exchanges are consistent across different experiments. This is expected for a disk bandwidth limited workload. However, the measured rates of memory page exchanges are surprising: we expected to see around 3250 page/s (13,000 KB/s divided by 4 KB memory pages should produce $\approx$ 3250 page/s), but we observe rates of memory page exchanges 8 times higher. The explanation is that the block size at a "raw" disk device level is 512 bytes. Thus each 4 KB memory page is used for transferring only 512 bytes of data. This leads to rates of memory page exchanges 8 times higher and, as a result, to a significantly higher CPU overhead in *Domain0*.

File Size	Transfer Time (sec)	Tput (MB/s)	Domain0 CPU Usage (%)	Memory Page Exch. (pages/s)
0.5 GB	38.4 sec	13.0 MB/s	12.68%	27,342
1 GB	76.8 sec	13.0 MB/s	12.5%	27,240
5 GB	379.5 sec	13.2 MB/s	12.52%	27,508
10 GB	763.6 sec	13.1 MB/s	12.51%	27,254

Table 1: Summary of "raw" disk performance measurements.

We repeated the same set of experiments for a disk device that is mounted as a file system. Table 2 summarizes the results collected for the new set of experiments. The transfer time is practically unchanged and again is directly proportional to the transferred amount of data. The achieved disk bandwidth is similar to the first set of experiments. However,

File Size	Transfer Time (sec)	Tput (MB/s)	Domain0 CPU Usage (%)	Memory Page Exch. (pages/s)
0.5 GB	38.3 sec	13.0 MB/s	4.08%	3,275
1 GB	77.8 sec	12.8 MB/s	4.1%	3,387
5 GB	386.3 sec	12.9 MB/s	3.98%	3384
10 GB	772.1 sec	13. MB/s	3.91%	3383

Table 2: Measurements for a disk device mounted as a file system.

the *Domain0* CPU usage and the rates of memory page exchanges are much lower than for the first set of experiments. The measured rates of memory page exchanges are close to our initial expectations of 3250 page/s. The measured CPU usage in *Domain0* is only about 4% for all the experiments in the second set, and it correlates well with the rates of memory page exchanges.

To quantify the overhead introduced by our instrumentation and performance monitor, we repeated all the experiments for the original Xen 2.0. The performance results for the instrumented and non-instrumented, original version of Xen 2.0 are practically indistinguishable.

In our future work, we intend to design a set of resource allocation policies that take this CPU overhead into account.

References

[1] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. Proc. of ACM SOSP, October 2003.

[2] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, M. Williamson. Reconstructing I/O. Tech. Report, UCAM-CL-TR-596, August 2004.

[3] M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton, F. Gittler. SoftUDC: A Software-based data center for utility computing. IEEE Computer, Special issue on Internet Data Centers, pp. 46-54, November, 2004.

[4] D. Mosberger, T. Jin. Httperf—A Tool for Measuring Web Server Performance. Proc. of Workshop on Internet Server Performance, 1998.

[5] F. Prefect, L. Doan, S.Gold, and W. Wilcke. Performance Limiting Factors in Http (Web) Server Operations. Proc. of COMPCON'96, Santa Clara, 1996, pp.267-273.

[6] The Workload for the SPECweb96 Benchmark. `http://www.specbench.org/osg/web96/workload.html`

[7] The Workload for the SPECweb99 Benchmark. `http://www.specbench.org/osg/web99/workload.html`

[8] A. Whitaker, M. Shaw, and S. Gribble. Denali: A Scalable Isolation Kernel. Proc. of ACM SIGOPS European Workshop, September 2002.