XenMon: QoS Monitoring and Performance Profiling Tool

Diwaker Gupta, Rob Gardner, Ludmila Cherkasova
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2005-187
October 18, 2005*

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

The goal of this short paper is twofold: 1) it briefly describes a new performance monitoring tool, XenMon, that we built for the Xen-based virtual environment, and 2) it presents a performance case study that demonstrates and explains how different metrics reported by XenMon can be used in gaining insight into an application's performance and its resource usage/requirements, especially in the case of I/O intensive applications.

* Internal Accession Date Only
Approved for External Publication
© Copyright 2005 Hewlett-Packard Development Company, L.P.

XenMon: QoS Monitoring and Performance Profiling Tool

Performance Study: How Much CPU Needs to Be Allocated to Dom_O
for Efficient Support of Web Server Applications?

Diwaker Gupta, Rob Gardner, Ludmila Cherkasova
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94303
e-mail: dgupta@cs.ucsd.edu [*], rob.gardner@hp.com, cherkasova@hpl.hp.com

Abstract. *The goal of this short paper is twofold: 1) it briefly describes a new performance monitoring tool, XenMon, that we built for the Xen-based virtual environment, and 2) it presents a performance case study that demonstrates and explains how different metrics reported by XenMon can be used in gaining insight into an application's performance and its resource usage/requirements, especially in the case of I/O intensive applications.*

1 Introduction

Virtual Machine Monitors (VMMs) are gaining popularity in enterprise environments as a software-based solution for building shared hardware infrastructures via virtualization. Virtualization technology is a key component in creating a more agile, dynamic infrastructure. For large enterprises it offers an ideal solution for server and application consolidation in an on-demand utility. Forrester Research estimates that businesses generally end up using somewhere between 8 and 20 percent of the server capacity they have purchased. Virtualization technology will help to achieve greater system utilization while lowering total cost of ownership and responding more effectively to changing business conditions.

As virtual machines enter the mainstream and are deployed in larger numbers, manageability, automation, accurate resource accounting, performance isolation, QoS-aware resource allocation, and billing will be the real differentiators to help customers create dynamic infrastructures. The Xen virtual machine monitor [1, 2] is Open Source software that allows multiple operating systems to execute concurrently and efficiently on commodity x86 hardware [1].

For supporting the resource allocation and management functions, we implemented an accurate monitoring and performance profiling infrastructure, called XenMon, that reports the resource usage of different VMs and provides an additional insight into shared resource access and resource scheduling in Xen.

Such a monitoring system is broadly needed for assistance in billing and for a wide variety of management tasks such as: i) support for policy-based resource allocation; ii) admission control of new VMs; iii) support for VMs migration; iv) QoS provisioning of VMs. This resource monitoring infrastructure forms a foundation for higher-level value-added services with QoS guarantees.

This paper briefly describes the new performance monitoring tool, XenMon, that we built for Xen, and presents a performance case study that demonstrates and explains how different metrics reported by Xen-

[*]Diwaker Gupta worked at HPLabs during the summer of 2005, and currently is back at UCSD to finish his PhD.
[1]In this paper, we do not describe the Xen architecture. For description we refer readers to the original Xen papers [1, 2].

Mon can be used in gaining insight into an application's performance and its resource usage/requirements, especially in the case of I/O intensive applications.

Our performance study looks at the question: how sensitive is the performance of I/O intensive applications (in particular a web server) to the amount of CPU allocated to Dom_0 where Dom_0 is a domain that hosts the device drivers and that is a critical service domain for I/O intensive applications .

This paper is organized as follows. Sections 2 and 3 briefly present the architecture of XenMon and the main metrics reported by XenMon. Sections 4 and 5 describe the experimental setup and application used in our performance study and the four resource allocation configurations which are used in the study. Sections 6 - 8 in a step-by-step manner present reported metrics for all four configurations, explain their meanings, and interpret them with respect to application behavior and performance. Appendix A contains the "README" file for XenMon.

2 XenMon

In this section, we briefly describe the architecture of XenMon. There are three main components in XenMon (Figure 1):

- xentrace: this is a light weight event generation facility present in Xen. Using xentrace it is possible to raise events at arbitrary control points in the hypervisor. Xentrace allows some attributes to be associated with each event (for instance, for a "domain scheduled" event, the associated attributes might be the ID of the scheduled domain and the timestamp of the event).
- xenbaked: Note that the event stream generated by xentrace is not very useful by itself. Xenbaked is a user-space process that catches events generated by xentrace and processes them into meaningful information. For instance, we might collate and aggregate domain sleep and wake events to determine the time for which a domain was blocked in a given interval.
- xenmon: this is the front-end for displaying and logging the data.

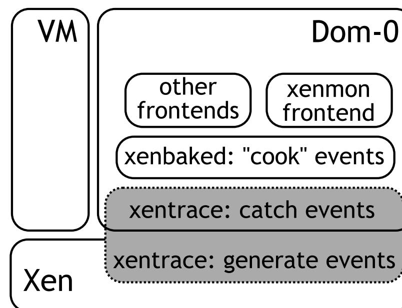

Figure 1: XenMon Architecture

We would like to emphasize some salient features of this architecture:

- xenbaked is highly configurable – one can specify how many samples to record per second, how many seconds of history should be kept, the sampling frequency etc. Further, xenbaked exports the processed data in a language neutral manner so that front ends can be written in any language. Currently this is done via a read only shared memory region, which means that multiple front-ends might read and display the data using the same xenbaked instance. We are currently exploring making this interface network transparent via XML-RPC.

- xenmon is written in python which makes it very portable. Currently xenmon provides a curses based front end for online viewing of data, as well as logging to files for post processing.
- xenmon is low overhead – we observe a maximum of 1-2% overhead in our experiments. Since the data exposed by xenbaked is language neutral, it is very easy to write faster frontends (say in C) if this overhead is unacceptable.

3 Metrics

XenMon reports a variety of metrics that are accumulated over the *i)* execution period (time interval that domain was scheduled to use CPU); *ii)* last second, and *iii)* 10 seconds.

There is a group of metrics that present how the time is spent by each domain:

- *CPU usage* – shows a percentage of time (for example, over 1 second) when a domain used CPU;
- *blocked time* – reflects a percentage of time a domain spent while blocked on some I/O event, i.e. a domain was not on the run queue;
- *waiting time* – shows a percentage of time a domain spent waiting on the run queue to be scheduled for CPU.

These three main metrics completely cover how time is spent by a domain.

XenMon reports a metric *execution count* that reflects how often a domain has been scheduled on a CPU during the measurement period (e.g. 1 second). When a reading of this metric is combined with CPU usage/waiting time per execution period – it provides an insight into the scheduler behavior.

For an I/O intensive application, XenMon provides *I/O count metric* that is a rough measure of I/O requested by the domain. It is the number of memory page exchanges (or page "flips") between a domain and Dom_0. The number of pages exchanged may not accurately reflect the number of bytes transferred to/from a domain due to partial pages being used by the network protocols, etc. But it does give a good sense of the magnitude of I/O being requested by a domain. We will explain this metric in more detail in Section 7.

4 Experimental Setup and Four Configurations Under 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* (which we denote Dom_0), 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. Dom_0 also may host unmodified Linux device drivers and play the role of a driver domain. In our experimental setup, we use Dom_0 as a driver domain as shown in FigureXen2.

Thus, for I/O intensive applications, CPU usage has two components: CPU consumed by the guest virtual machine (VM), where the application resides, and CPU consumed by Dom_0 that incorporates device driver and performs I/O processing on behalf of the guest domain. We performed a sensitivity study of how web server performance depends on the amount of CPU allocated to Dom_0.

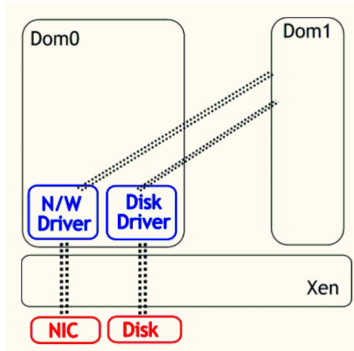

Figure 2: Experimental Setup.

In the performance study below, we used a 1-CPU HP x4000 Workstation with a 1.7 GHz Intel Xeon processor, 2 GB RAM. We ran Xen 3.0. Dom_1 ran Apache HTTP server version 2.0.40. We used the *httperf* tool [3] for sending the client requests that retrieve 10 KB files. 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 1 Gbit/s network.

Xen 3.0 was booted with the Borrowed Virtual Time scheduler (BVT) as a CPU scheduler. BVT provides proportional fair share CPU scheduling based on weights. Each runnable domain receives a share of CPU in proportion to its weight. Current BVT implementation is a work conserving scheduler, i.e. if one domain has "no work to do" then the other domain can get the entire CPU for its execution (independent of the domain's weight). Thus, in the current BVT scheduler, when two different domains are assigned equal weights it does not mean that each of them will get 50% of CPU resources. It only means that these domains are entitled to get the same amount of CPU when both of them have work to do.

We aim to answer the question: how sensitive is web server performance to different amounts of CPU allocated to Dom_0 that hosts device drivers.

We designed 4 different configurations where we varied the CPU amount allocated to Dom_0 relatively to Dom_1:

- **Conf_0.5**: Dom_0 is assigned a weight of 20, and Dom_1 is assigned a weight of 10. This means that Dom_0 is allocated half of the CPU share compared to Dom_1 CPU share, i.e. $Dom_0 = 0.5 \times Dom_1$ in CPU shares;
- **Conf_Equal**: Dom_0 and Dom_1 are assigned equal weights of 10 to get the same CPU share;
- **Conf_2**: Dom_0 is assigned a weight of 5, and Dom_1 is assigned a weight of 10. This means that that Dom_0 is allocated twice as much CPU compared to Dom_1 CPU share, i.e. $Dom_0 = 2 \times Dom_1$ in CPU shares;
- **Conf_10**: Dom_0 is assigned a weight of 1, and Dom_1 is assigned a weight of 10. This means that that Dom_0 is allocated ten times as much CPU compared to Dom_1 CPU share, i.e. $Dom_0 = 10 \times Dom_1$ in CPU shares.

5 Web Server Performance

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.

Figure 3 a) shows achievable web server throughput under the four CPU allocation configurations described above. The results reflect a specific trend in a web server performance under the increased CPU share to Dom_0.

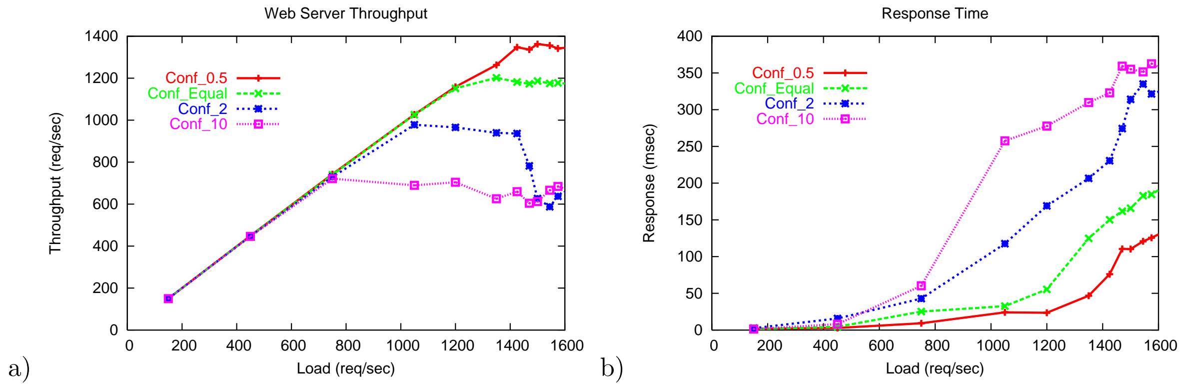

Figure 3: a) Web server throughput; b) Web server response time.

Interestingly enough, web server throughput is best under configuration *Conf_0.5* when Dom_0 is allocated a half amount of CPU that is allocated to Dom_1. Under this configuration web server throughput reaches 1380 request/sec.

The common trend observed in Figures 3 a) is that a larger CPU share allocated to Dom_0 leads to worse web server throughput: under configuration *Conf_Equal* web server throughput is 1200 request/sec, for configuration *Conf_2* web server throughput is 980 request/sec, and it reaches only 700 request/sec under configuration *Conf_10*.

Web server response time for the four configurations is shown in Figure 3 b). It supports the same trend: web server response is best under configuration *Conf_0.5* when Dom_0 is allocated half the amount of CPU that is allocated to Dom_1.

It seems to be counterintuitive. Why does a higher CPU share allocated to Dom_0 lead to worse web server performance? How can it be explained?

6 Where Does Time Go? Three Main XenMon Metrics: CPU Usage, Blocked and Waiting Time

First of all, let us see how CPU usage is impacted by different weight assignments to Dom_0. Figure 4 shows CPU utilization for Dom_0 and Dom_1 under the four different configurations. CPU usage increases for both domains under increased load to a web server.

Figure 4 a) shows that under configuration *Conf_0.5* and heavy load to a web server, Dom_0 uses 30% of CPU while Dom_1 gets 68% of CPU (there is about 2% CPU overhead due to XenMon tool usage). It is the only configuration where the targeted CPU allocation (1:2 ratio for $Dom_0 : Dom_1$ CPU allocation ratio) is really achieved.

For the other configurations, with assignment of increased CPU share to Dom_0, we can see that CPU

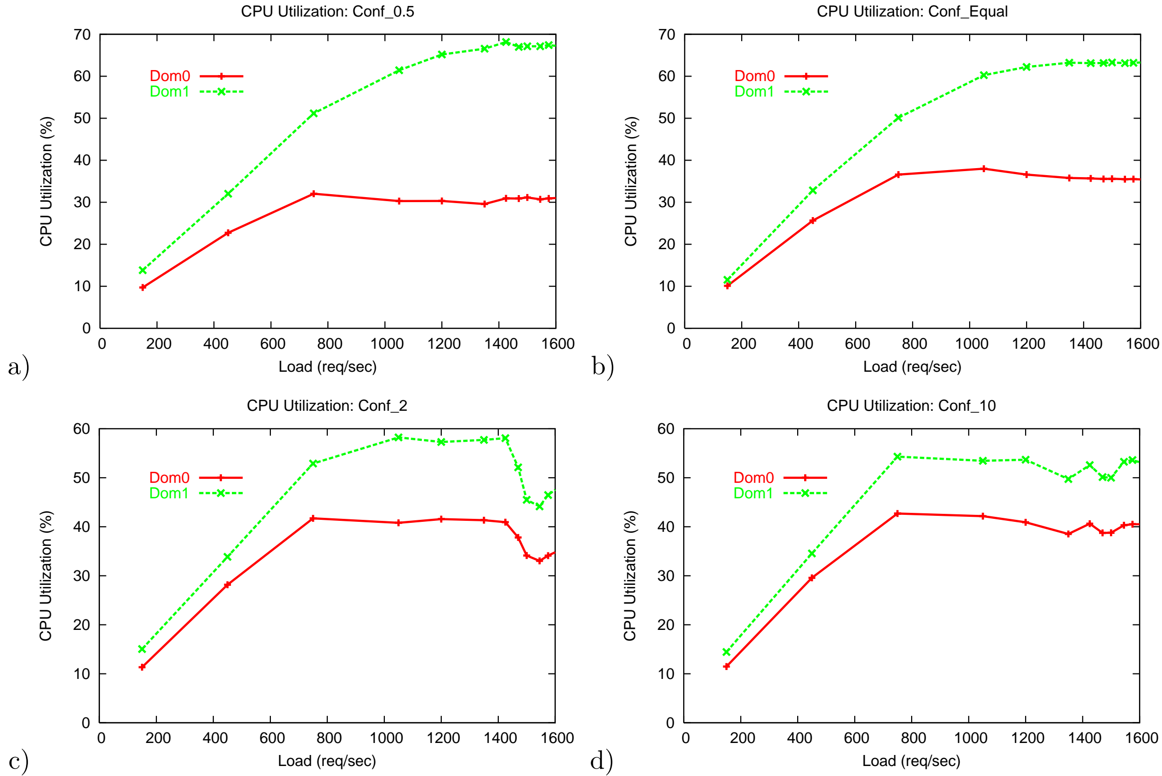

Figure 4: CPU Utilization under BVT scheduler with different weights: a) *Conf_0.5*; b) *Conf_Equal*; c) *Conf_2*; d) *Conf_10*.

usage by Dom_0 increases as shown in Figures 4 b), c) and d), but it does not go above 45% for configuration *Conf_10*, where Dom_0 is allocated ten times as much CPU compared to Dom_1 CPU share. At the same time, this increased CPU consumption by Dom_0 leave less CPU available to Dom_1: under configuration *Conf_10*, Dom_1 gets only 55% of CPU compared to 68% of CPU used by Dom_1 in configuration *Conf_0.5*. Clearly, this redistribution in CPU usage provides a first insight why web server throughput decreases so dramatically under increased CPU allocation to Dom_0.

Figure 5 shows percentage of blocked time for Dom_0 and Dom_1 under the four different configurations. The blocked time metric represents a percentage of time when a domain is blocked on I/O events.

Figure 5 a) shows the percentage of blocked time by Dom_0 and Dom_1 under configuration *Conf_0.5*. It is the only configuration in the set of four configurations under study, where Dom_1 has a higher blocked time percentage than Dom_0. Intuitively, in configuration *Conf_0.5*, Dom_1 is typically blocked as much as Dom_0 waiting on the I/O events to be processed by Dom_0. It is a logical scenario for an I/O intensive and interrupt driven application like web server.

What is happening in the system under increased CPU allocation to Dom_0?

As shown in Figures 5 b), c) and d), under increased CPU allocation to Dom_0, blocked time of Dom_1 becomes smaller: Dom_0 services interrupts much faster and Dom_1 running a web server seldomly blocks on I/O. Under higher CPU allocation to Dom_0, the blocked time for Dom_0 increased significantly.

It seems a bit strange at first glance, but a complementary waiting time metric discussed below helps in understanding the system behavior in more detail.

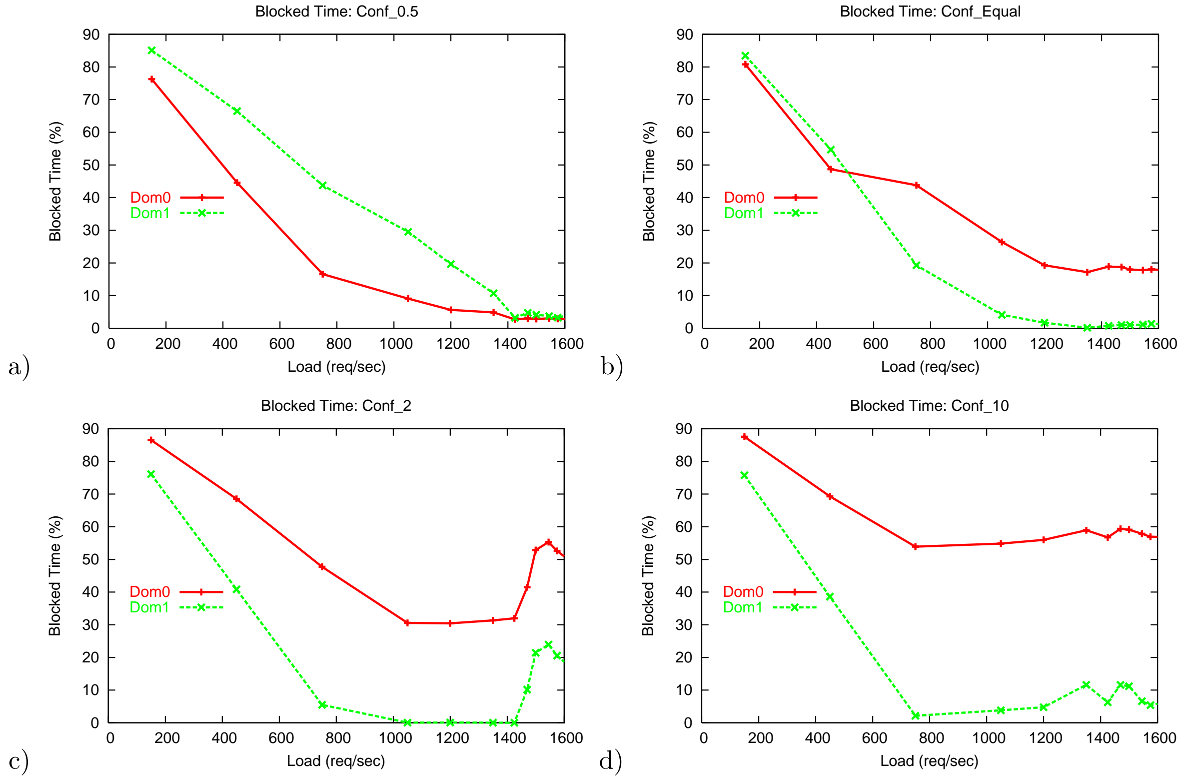

Figure 5: Blocked Time under BVT scheduler with different weights: a) *Conf_0.5*; b) *Conf_Equal*; c) *Conf_2*; d) *Conf_10*.

Figure 6 shows the percentage of waiting time for Dom_0 and Dom_1 under the four different configurations. The waiting time metric represents a percentage of time when a domain becomes runnable and is waiting for CPU on the run queue.

Figure 6 a) shows the percentage of waiting time for Dom_0 and Dom_1 under configuration *Conf_0.5* Interestingly, it is the only configuration in the set of four configurations under study where Dom_0 has a significantly higher waiting time than Dom_1.

In configuration *Conf_Equal* shown in Figure 6 b), waiting time for Dom_0 decreases while waiting time for Dom_1 is getting larger. However, for most of the points, waiting time for Dom_0 is still higher than for Dom_1.

For configurations *Conf_2* and *Conf_10*, the situation changes dramatically: under *Conf_10*, waiting time for Dom_0 is only about 2% while for Dom_1 it increases to 40%. Thus, when Dom_0 is allocated a higher CPU share compared to Dom_1, it results that Dom_0 experiences a much shorter waiting time in the run queue and faster access to CPU.

Here, it might be worthwhile to discuss some additional details of how the BVT scheduler operates. The classic BVT scheduler is based on the *virtual time* concept, dispatching the runnable thread/virtual machine with the *earliest virtual time* for CPU access first.

Each runnable domain Dom_i receives a share of CPU in proportion to its weight $weight_i$. To achieve this, the virtual time of the currently running Dom_i is incremented by its running time divided by $weight_i$. This way, the virtual time of domains with higher CPU allocations increases more slowly, and such a domain has a higher chances in getting CPU allocated to it because its virtual time is lower.

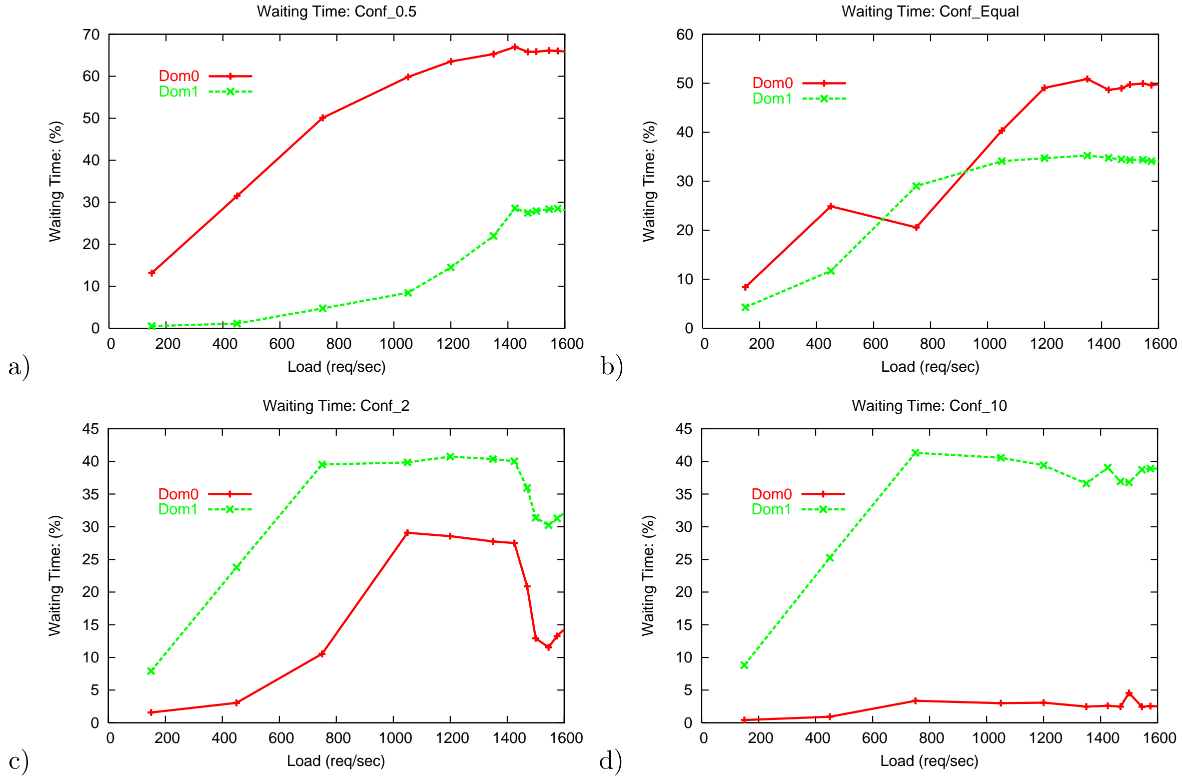

Figure 6: Waiting Time under BVT scheduler with different weights: a) *Conf_0.5*; b) *Conf_Equal*; c) *Conf_2*; d) *Conf_10*.

This explains how Dom_0 gets "priority" treatment under configuration *Conf_10*, where a high CPU share allocated to Dom_0 results in a smaller virtual time for Dom_0 and its ability to get CPU faster once it is on the run queue. In its own turn, it results that Dom_0 gets served quickly, and after that it gets blocked on the next I/O event. This is why with the higher CPU allocation to Dom_0, we have seen an increased blocked time for Dom_0 (see Figures 5 d)).

For Dom_1, the situation is somewhat reversed. When Dom_0 is allocated a higher CPU share than Dom_1, Dom_1 gets blocked on I/O events less and less often (since they are processed by Dom_0 in faster manner). In fact, under heavier load Dom_1 never gets blocked and always has work to do. We can see this through the increased waiting time metric. Practically, under heavier load, the time for Dom_1 consists of two components: waiting time in the run queue and CPU processing (run) time.

Now, we have all the components showing how the time is spent by each domain in the four configurations under study. However, it is still unclear why web server throughput suffers so much under configurations where Dom_0 is allocated a higher CPU share compared to Dom_1, and why web server performance is so drastically better under configuration *Conf_0.5*.

The following section helps to get a good insight of the details of I/O processing under the four configurations in our study.

7 How Efficient is I/O Processing: XenMon I/O Count Metric

A few words about I/O support in Xen. 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.

Thus, in order to account for different I/O related activities in Dom_0 (that "hosts" the unmodified device drivers), XenMon observes the memory page exchanges between Dom_0 and Dom_i. XenMon computes two related metrics: it measures the number of memory page exchanges performed per second,and it computes an average number of memory page exchanges performed per execution period (averaging it over one second). For brevity, XenMon uses the term I/O count for the number of memory page exchanges.

Figure 7 a) shows the I/O count per second between Dom_0 and Dom_1. It clearly reflects a trend that under increased CPU allocation to Dom_0 the overall I/O count per second decreases.

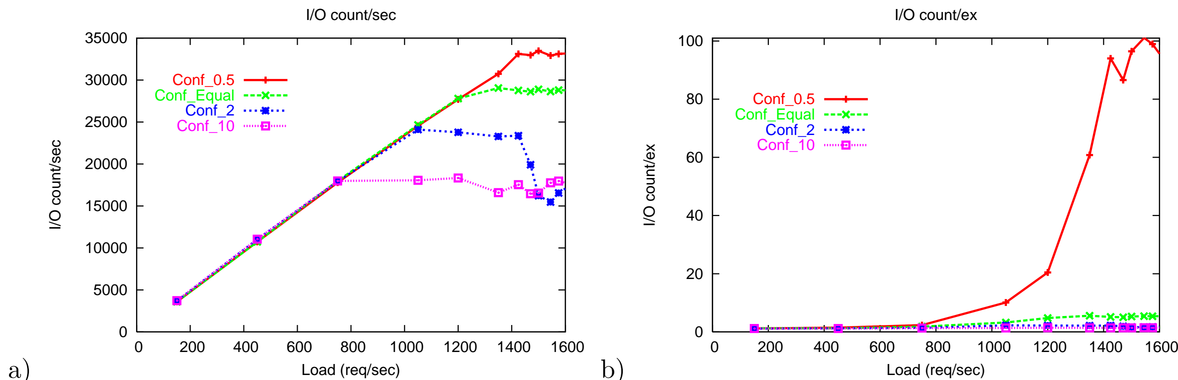

Figure 7: a) I/O count (memory page flips) per second; b) I/O count (memory page flips) per execution period.

Figure 7 b) shows the I/O count between Dom_0 and Dom_1 per execution period in Dom_0. This figure offers a very interesting explanation of what happens in the system under increased CPU allocation to Dom_0. In configuration *Conf_10*, there are fewer than 1.4 memory page exchanges per execution period on average. Such behavior persists even when the web server is experiencing a heavy load. Practically, it means that Dom_0 is scheduled for CPU processing on each interrupt! In such a way, the interrupt processing becomes very expensive and has a high processing cost since it includes domain switch overhead. This leads to a lower overall I/O count per second between Dom_0 and Dom_1. Under configuration *Conf_10*, I/O count per second is only half of I/O count under configuration *Conf_0.5*. This observation is highly correlated with trends in web server throughput: under configuration *Conf_10*, web server throughput is 700 requests per second, that is half of web server throughput under configuration *Conf_0.5*, where web server throughput reaches 1380 requests per second.

Under configuration *Conf_0.5*, the interrupt processing is strikingly efficient: there are up to 100 memory page exchanges per execution period when a web server is under heavy load. Such performance is tightly coupled with BVT scheduling decisions under different weight assignments to Dom_0. We discuss this phenomena in the next section.

8 Scheduling Mystery: Number of Execution Periods and Their Duration Reported by XenMon

Figure 8 shows the number of execution periods under the four different configurations.

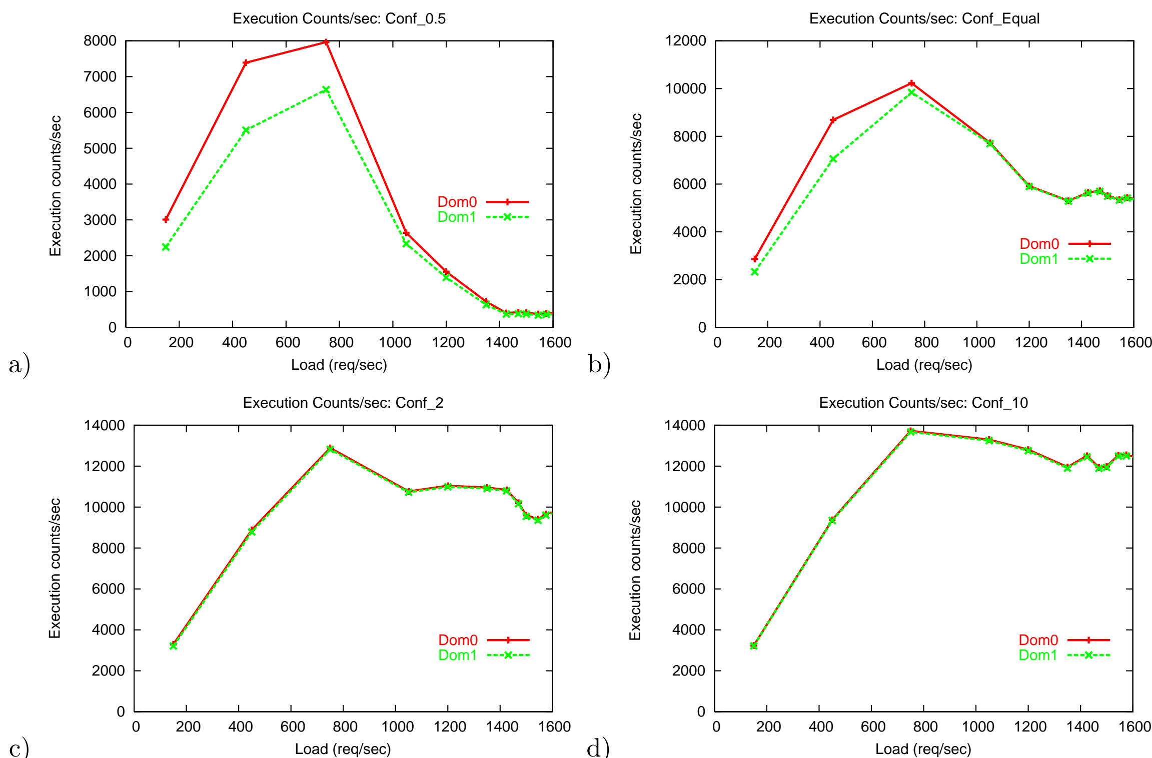

Figure 8: Number of execution periods per second under BVT scheduler with different weights: a) *Conf_0.5*; b) *Conf_Equal*; c) *Conf_2*; d) *Conf_10*.

Let us more carefully analyze the XenMon results for configuration *Conf_0.5* shown in Figure 8 a). Under a relatively light load to the web server, there is a high number (up to 7,000 - 8,000) of execution periods per second for both domains Dom_0 and Dom_1.

Figure 9 a) shows the average CPU time received per execution period under configuration *Conf_0.5*. Practically, this metric reflects the average duration of each execution period. In particular, XenMon reveals that under a light load to the web server (less than or equal to 800 requests per second), there is a high number of "short" execution periods per second with an average duration of less than 60 microseconds.

However, under heavier load to the web server, in configuration *Conf_0.5*, there is efficient "aggregation" of interrupts and "work to do" in Dom_0, that results in an efficient aggregation of "work to do" for Dom_1. This leads to a much smaller number (around 400) of "longer" execution periods with an average duration of 1,000 microseconds for Dom_0 and 2,000 microseconds for Dom_1. Clearly, over such longer execution periods, Dom_0 is capable of processing an order of magnitude more I/O interrupts with much less processing cost. Under this configuration, the targeted CPU allocation (1:2 ratio for $Dom_0 : Dom_1$ CPU allocation ratio) is truly achieved.

As for configurations with increased CPU allocation to Dom_0 such as *Conf_2* and *Conf_10*, XenMon reveals that even under heavy load to the web server, these configurations are causing a high number (10,000 -12,000) of "short" execution periods with an average duration of 35-50 microseconds as shown

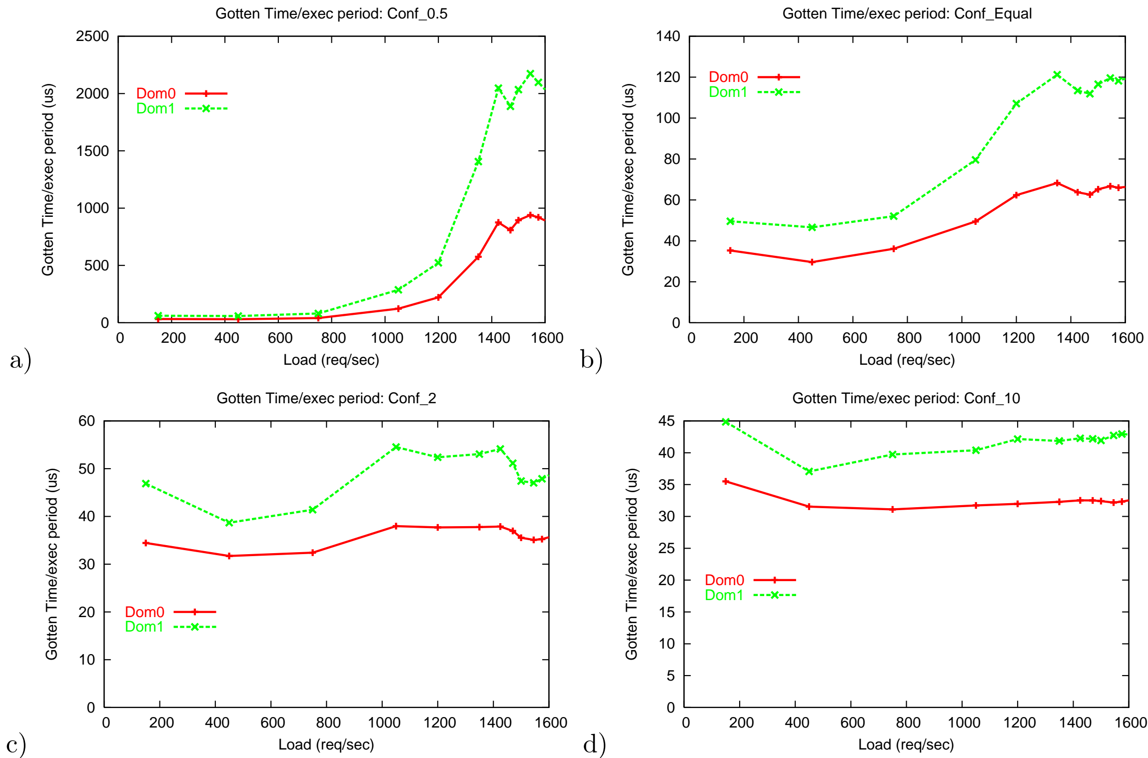

Figure 9: Gotten time per execution period under BVT scheduler with different weights: a) *Conf_0.5*; b) *Conf_Equal*; c) *Conf_2*; d) *Conf_10*.

in Figures 8 c), d) and Figures 9 c), d). Such frequent domain switches for CPU processing results in high overhead for I/O processing and leads to poor application performance.

9 Conclusion

In this short paper, we described XenMon's monitoring and profiling capabilities that can be used in gaining insight into an application's performance and its resource usage/requirements (especially, in the case of I/O intensive applications). We performed a sensitivity study of web server performance based on the amount of CPU allocated to Dom_0. The web server exhibited its best performance when the ratio of CPU share of Dom_0 to Dom_1 was set to $1 : 2$. Surprisingly, with increased CPU allocation to Dom_0, web server performance decreased significantly.

XenMon helped in revealing the system behavior and in getting insight into how I/O processing impacts the application performance.

We believe that XenMon with its profiling capabilities at the level of execution periods will also be useful in understanding the benefits and shortcomings of different CPU schedulers available in Xen.

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

A README file for XenMon

```
XenMon: Xen Performance Monitor
-----------------------

The XenMon tool makes use of the existing xen tracing feature to provide fine
grained reporting of various domain related metrics. It should be stressed that
the xenmon.py script included here is just an example of the data that may be
displayed. The xenbake demon keeps a large amount of history in a shared memory
area that may be accessed by tools such as XenMon.

For each domain, XenMon reports various metrics. One part of the display is a
group of metrics that have been accumulated over the last second, while another
part of the display shows data measured over 10 seconds. Other measurement
intervals are possible, but we have just chosen 1s and 10s as an example.


Execution Count
---------------
 o The number of times that a domain was scheduled to run (ie, dispatched) over
 the measurement interval


CPU usage
---------
 o Total time used over the measurement interval
 o Usage expressed as a percentage of the measurement interval
 o Average cpu time used during each execution of the domain


Waiting time
------------
This is how much time the domain spent waiting to run, or put another way, the
amount of time the domain spent in the "runnable" state (or on the run queue)
but not actually running. XenMon displays:

 o Total time waiting over the measurement interval
 o Wait time expressed as a percentage of the measurement interval
 o Average waiting time for each execution of the domain

Blocked time
------------
This is how much time the domain spent blocked (or sleeping); Put another way,
the amount of time the domain spent not needing/wanting the cpu because it was
waiting for some event (ie, I/O). XenMon reports:

 o Total time blocked over the measurement interval
 o Blocked time expressed as a percentage of the measurement interval
 o Blocked time per I/O (see I/O count below)
```

Allocation time

This is how much cpu time was allocated to the domain by the scheduler; This is distinct from cpu usage since the "time slice" given to a domain is frequently cut short for one reason or another, ie, the domain requests I/O and blocks. XenMon reports:

 o Average allocation time per execution (ie, time slice)
 o Min and Max allocation times

I/O Count

This is a rough measure of I/O requested by the domain. The number of page exchanges (or page "flips") between the domain and dom0 are counted. The number of pages exchanged may not accurately reflect the number of bytes transferred to/from a domain due to partial pages being used by the network protocols, etc. But it does give a good sense of the magnitude of I/O being requested by a domain. XenMon reports:

 o Total number of page exchanges during the measurement interval
 o Average number of page exchanges per execution of the domain

Usage Notes and issues

 - Start XenMon by simply running xenmon.py; The xenbake demon is started and stopped automatically by XenMon.
 - To see the various options for XenMon, run xenmon.py -h. Ditto for xenbaked
 - XenMon also has an option (-n) to output log data to a file instead of the curses interface
 - NDOMAINS is defined to be 32, but can be changed by recompiling xenbaked
 - xenmon.py appears to create 1-2% cpu overhead; Part of this is just the overhead of the python interpreter. Part of it may be the number of trace records being generated. The number of trace records generated can be limited by setting the trace mask (with a dom0 Op), which controls which events cause a trace record to be emitted.
 - To exit XenMon, type 'q'
 - To cycle the display to other physical cpu's, type 'c'

Future Work
----
o RPC interface to allow external entities to programmatically access processed data
o I/O Count batching to reduce number of trace records generated

Authors

Diwaker Gupta <diwaker.gupta@hp.com>
Rob Gardner <rob.gardner@hp.com>
Lucy Cherkasova <lucy.cherkasova.hp.com>