When Virtual is Harder than Real: Resource Allocation Challenges in Virtual Machine Based IT Environments

Ludmila Cherkasova, Diwaker Gupta[1], Amin Vahdat[1]
Enterprise Systems and Software Laboratory
HP Laboratories Palo Alto
HPL-2007-25
February 8, 2007*

virtualization, Xen, resource allocation, benchmarks, application performance, CPU schedulers

The primary motivation for enterprises to adopt virtualization technologies is the promise of creating a more agile and dynamic IT infrastructure — with server consolidation, high resource utilization, the ability to quickly add and adjust capacity on demand — while lowering total cost of ownership and responding more effectively to changing business conditions. However, effective management of virtualized IT environments introduces new and unique requirements such as dynamically resizing and migrating virtual machines in response to changing application demands. Such capacity management methods should work in ensemble with the underlying resource management mechanisms. Using Xen and its three different CPU schedulers, we analyze the impact of the choice of scheduler and its parameters on application performance and discuss challenges in estimating the application resource requirements in virtualized environments.

* Internal Accession Date Only
[1]University of California, San Diego
Approved for External Publication

When Virtual is Harder than Real: Resource Allocation Challenges in Virtual Machine Based IT Environments

Ludmila Cherkasova
Hewlett-Packard Laboratories
{lucy.cherkasova@hp.com}

Diwaker Gupta Amin Vahdat
University of California, San Diego
{dgupta,vahdat}@cs.ucsd.edu

Abstract

The primary motivation for enterprises to adopt virtualization technologies is the promise of creating a more agile and dynamic IT infrastructure — with server consolidation, high resource utilization, the ability to quickly add and adjust capacity on demand — while lowering total cost of ownership and responding more effectively to changing business conditions. However, effective management of virtualized IT environments introduces new and unique requirements such as dynamically resizing and migrating virtual machines in response to changing application demands. Such capacity management methods should work in ensemble with the underlying resource management mechanisms. Using Xen and its three different CPU schedulers, we analyze the impact of the choice of scheduler and its parameters on application perfomance and discuss challenges in estimating the application resource requirements in virtualized environments.

1 Introduction

Virtualization is emerging as a key mechanism of scaling the IT infrastructure and enabling enterprises to move from vertical silos of servers to horizontal pools of resources. Server virtualization provides the ability to slice larger, underutilized physical servers into smaller, virtual ones. Although virtualization has been around for more than three decades, it has found its way into the mainstream only recently, as a consequence of the recent developments in virtualization software and improved hardware support. A variety of solutions — both commercial and open source — are now available for commodity systems.

The motivation for enterprises to adopt virtualization technologies is increased flexibility, the ability to quickly re-purpose server capacity to better meet the needs of application workload owners, and to reduce overall costs of ownership. Virtualization services offer interfaces that support the life cycle management (e.g., create, destroy, move, size capacity) of VMs that are provided with access to shares of resource capacity (e.g., cpu, memory, input-output). Furthermore, some virtualization mechanisms provide the ability to dynamically migrate VMs from one physical machine to another without interrupting application execution. For large enterprises it offers an ideal solution for server and application consolidation.

Unfortunately, the complexity of these virtualized environments presents additional management challenges. Garfunkel and Rosenblum discussed security challenges in virtualized environments [9]. In this work, we address resource allocation and capacity management problems in VM-based environments. There are many workloads, a finite number can be hosted by each server, and each workload has capacity requirements that may frequently change based on business needs. While VMs provide fault isolation, in an enterprise environment, it is also important that services receive performance and resource isolation, meaning that rogue services should not impact the performance of other applications that share the same infrastructure. Cost effective capacity management methods are not yet available. Moreover, such capacity management methods critically depend on the characteristics of the resource allocation mechanisms of the underlying VM platform.

While our broader premise is that resource allocation for VMs is in general a hard problem, in this paper we focus our attention on CPU scheduling. As a concrete example of the types of challenges involved, we analyze the CPU schedulers in the Xen VMM [6] in the context of traditional workload managers. Workload managers [2, 4] and similar tools were until a few years ago known only to mainframers and users of large Unix environments. These technologies have their own requirements from the underlying resource management mechanisms, e.g., CPU scheduling of VMs. Using Xen and its evolution with three different CPU schedulers, we demonstrate the challenges in choosing the appropriate scheduler features and parameters to support desirable application performance, as well as demonstrate the performance impact of these different choices.

2 CPU Schedulers for Virtual Machines

There are compelling reasons to use proportional share (PS) scheduling for CPU scheduling in VMs. PS scheduling allocates CPU in proportion to the number of shares (weights) that VMs have been assigned. Typically, PS schedulers are evaluated based on the level of **fairness**, i.e., the time interval over which the scheduler provides fair CPU allocation, and the allocation **error** which typically depends on the scheduler algorithm and its quantum size. An important distinction between fair-share schedulers and PS schedulers is the time granularity at which they operate. Proportional share schedulers aim to provide an instantaneous form of sharing among the active clients according to their weights. In contrast, fair-share schedulers attempt to provide a time-averaged form of proportional sharing based on the actual use measured over long time periods.

CPU schedulers can be further distinguished as supporting **work-conserving** (WC-mode) and/or **non work-conserving** (NWC-mode) modes. Under WC-mode, the shares are merely guarantees, and CPU is idle if and only if there is no runnable client. It means that in a case of two clients with equal weights and a situation when one of these clients is blocked, the other client can consume the entire CPU. With NWC-mode, the shares are caps, i.e., each client owns its fraction of the CPU. It means that in a case of two clients with equal weights, each client will get up to 50% of CPU, but the client will not be able to get more than 50% even if the rest of the CPU is idle.

We also distinguish **preemptive** and **non-preemptive** CPU schedulers. Preemptive schedulers rerun the scheduling decision whenever a new client becomes ready. If the new client has "priority" over the running client, the CPU preempts the running client and executes the new client. Non-preemptive scheduler only makes scheduling decisions when the running client voluntarily gives up CPU. Having a preemptive scheduler is important for achieving good performance of I/O intensive workloads in shared environment. These workloads are often blocked waiting for I/O events, and their performance could suffer in presence of CPU intensive jobs if the CPU scheduler is non-preemptive. However, choosing a right quantum size may alleviate this problem.

3 Workload Managers

A core requirement for effective virtualization is workload management, i.e., the ability to assign resources such as CPU, memory, and I/O to applications as precisely as possible. Workload management enables applications to provide service levels based on policies driven by time, price, and performance.

One simple approach for assigning CPU resources to VMs is static allocation. However, static allocation becomes inefficient under varying load: each VM must be sized to support the application's peak capacity requirements. Yet, most applications rarely need their peak amount. Workload managers aim to dynamically allocate resources to match application requirements.

The workload manager is layered upon a PS scheduler used in a NWC-mode. This mode provides performance isolation among workloads. Each VM receives its particular service rate regardless of whether any of the other VMs are using resources. Such isolation can be desirable in a shared environment for enterprise applications as it gives the appearance of dedicated access to resources. Adding new workloads to the pool has little impact on the performance behavior of workloads already in the pool.

Each resource container or virtual machine is pre-allocated specific shares of capacity for short time periods, e.g., 5 seconds. Then, based on the demands of the VM and the availability of resources, the allocations may be adjusted to ensure that each VM gets the capacity it needs. Since the decision of workload manager's controller is based on a difference between assigned and consumed CPU allocation, a scheduler with significant error in CPU allocation may cause unstable controller behavior. Thus, a prerequisite requirement to the underlying CPU scheduler is a small allocation error (typically, 1%-2%).

4 CPU Schedulers in Xen

Xen is unique among VM platforms because it allows users to choose among different CPU schedulers. But this choice comes with the burden of choosing the right scheduler and configuring it. Over the course of last three years, three different CPU schedulers were introduced, all allowing users specify CPU allocation via CPU shares (weights). Below, we briefly characterize their main features that at the time motivated their inclusion in Xen.

Borrowed Virtual Time (BVT) [7] is a fair-share scheduler based on the concept of *virtual time*, dispatching the runnable VM with the smallest virtual time first. Additionally, BVT provides low-latency support for real-time and interactive applications by allowing latency-sensitive client to "warp" back in virtual time to gain scheduling priority. The client effectively "borrows" virtual time from its future CPU allocation.

The scheduler is configured with a context switch allowance C, which is the *real time* by which the current VM is allowed to advance beyond another runnable VM with equal claim on the CPU (it is the basic time slice or time quantum of the algorithm). Each runnable domain 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 $weight_i$.

In summary, BVT has the following features:

- preemptive (if warp is used), WC-mode only;
- *optimally-fair*: the error between fair share and actual allocation is never greater than context switch

allowance C;

- low-overhead implementation on multiprocessors as well as uni-processors.

The lack of NWC-mode in BVT severely limited its usage, and led to the introduction of the next scheduler in Xen.

Simple Earliest Deadline First (SEDF) [11] uses real time-algorithms to ensure time guarantees. Each domain Dom_i specifies its CPU requirements by a tuple (s_i, p_i, x_i), where the *slice* s_i and the *period* p_i together represent the CPU share that Dom_i requests: Dom_i will receive at least s_i units of time in each period of length p_i. The boolean flag x_i indicates whether Dom_i is eligible to receive extra CPU time (WC-mode). This slack time is distributed in a fair manner after all the runnable domains received their CPU share. One can allocate 30% CPU to a domain by assigning either (3 ms, 10 ms, 0) or (30 ms, 100 ms, 0). The time granularity in the definition of the period impacts the scheduler fairness.

For each domain Dom_i, the scheduler keeps track of two additional values (d_i, r_i):

- d_i - time at which Dom_i's current period ends, also called the *deadline*. The runnable domain with earliest deadline is picked to be scheduled next;
- r_i - *remaining CPU time* of Dom_i in the current period.

In summary, SEDF has the following features:

- preemptive, WC and NWC modes;
- fairness depends on a value of the period.
- implements per CPU queue: this implementation lacks global load balancing on multiprocessors.

Credit Scheduler [1] is the latest PS scheduler in Xen featuring automatic load balancing of virtual CPUs across physical CPUs on an SMP host. Before a CPU goes idle, it will consider other CPUs to find any runnable VCPU. This approach guarantees that no CPU idles when there is runnable work in the system.

Each VM is assigned a *weight* and a *cap*. If the cap is 0 then VM can receive extra CPU (WC-mode). A non-zero cap (expressed as a percentage) limits the amount of CPU a VM receives (NWC-mode). The Credit scheduler uses 30 ms time slices for CPU allocation. A VM (VCPU) receives 30 ms before being preempted to run another VM. Once every 30 ms, the priorities (credits) of all runnable VMs are recalculated. The scheduler monitors resource usage every 10 ms. To some degree, Credit's computation of credits resembles virtual time computation in BVT. However, BVT has a context switch allowance C for defining a different size of the basic time slice (time quantum), and an additional low-latency support (via *warp*) for real-time applications.

In summary, Credit has the following features:

- non-preemptive, WC and NWC modes;
- global load balancing on multiprocessors.

In the next section, we present results of a performance study comparing these schedulers and their features in more detail.

5 Performance of CPU Schedulers in Xen

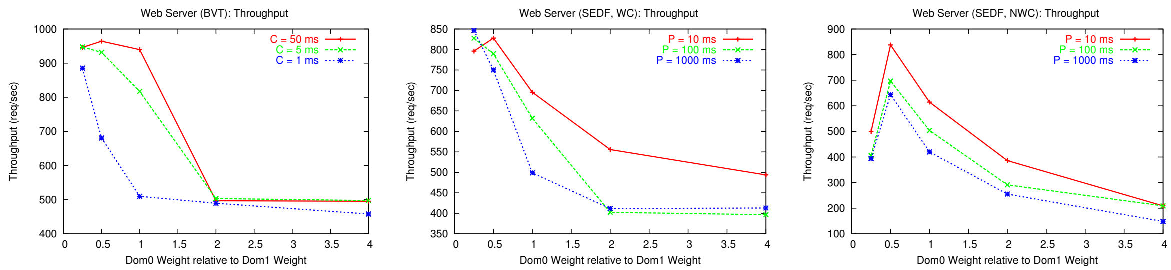
There are two popular I/O models for VMs, as demonstrated in the evolution of the I/O architecture of Xen. In its original design [6], the Xen VMM itself contained device driver code and provided safe, shared access for I/O hardware. Later, the Xen team proposed a new architecture [8] to allow unmodified device drivers to be hosted and executed in isolated "driver domains". Typically, the management domain Dom_0 hosts unmodified Linux device drivers and plays the role of the driver domain. This new I/O model results in a more complex CPU usage model. For I/O intensive applications, CPU usage has two components: CPU consumed by the guest virtual machine (VM) and CPU consumed by Dom_0 which hosts the device drivers and performs I/O processing on behalf of the guest domain.

Finding a satisfactory solution to the CPU allocation problem for applications executing in VMs requires answering several questions. How does one estimate the application CPU requirements and project them into two components: Dom_0 and guest domain's shares? How sensitive are I/O intensive applications to the amount of CPU allocated to Dom_0? Does allocation of a higher CPU share to Dom_0 mean a better performance for I/O intensive applications? How significant is the impact of scheduler parameters on application performance, e.g., context switch allowance C in BVT and period P_i in SEDF?

Further, additional functionality was the main motivation behind introducing new Xen schedulers. For example, SEDF added the NWC-mode missing in BVT, and Credit added automatic, transparent load balancing of VCPUs, missing in both BVT and SEDF. To the best of our knowledge, no one has done a thorough comparative performance evaluation of the different schedulers, so it is not immediately clear if configuring different schedulers with the same CPU allocation would result in similar application performance.

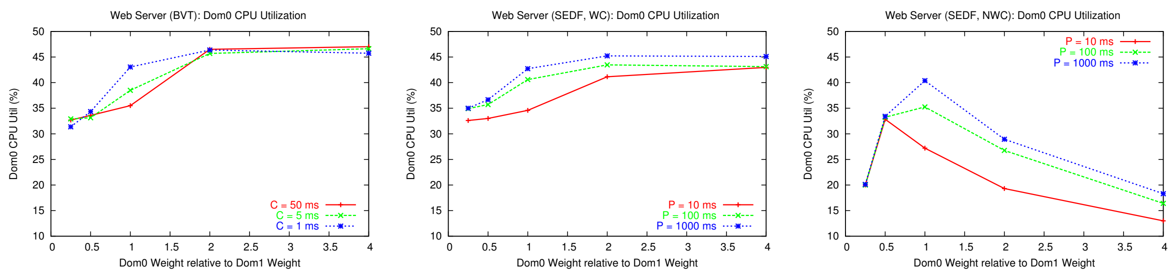
Our performance study aims to answer some of these questions. We performed experiments with three applications:

- **web server**: We measure web server throughput. In our workload, we request fixed size (10 KB) files using httperf [3].
- **iperf**: We measure maximum achievable network throughput using iperf [5].
- **disk read**: Finally, we benchmark disk read



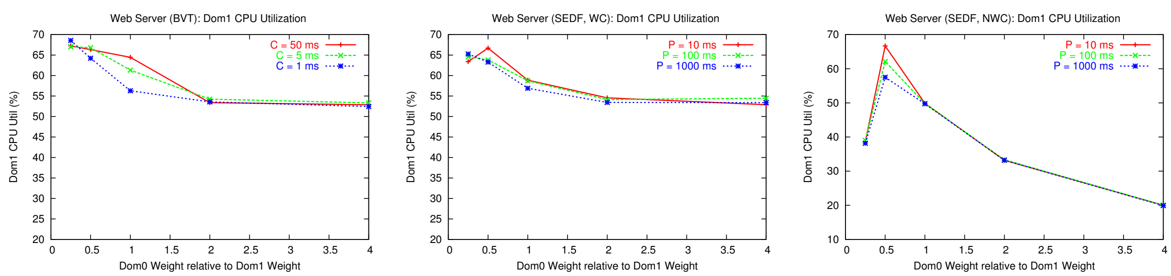
(a) BVT (WC-mode) (b) SEDF (WC-mode) (c) SEDF (NWC-mode)

Figure 1: Impact of schedulers' parameters on web server performance



(a) BVT (WC-mode) (b) SEDF (WC-mode) (c) SEDF (NWC-mode)

Figure 2: Dom_0's CPU usage under different schedulers' parameters



(a) BVT (WC-mode) (b) SEDF (WC-mode) (c) SEDF (NWC-mode)

Figure 3: Dom_1's CPU usage under different schedulers' parameters

throughput with the `dd` utility for reading 1000 1-KB blocks.

Our testbed consists of dual CPU HP workstations LP200R, with 1-GHz PIII processors, 2-GB RAM and 1-Gbit/s NICs running Xen 3.0.3. In this work, we present the results for single CPU configurations to separate comparison of the basic CPU scheduler properties from the load balancing issues for SMP configurations. We considered 5 different configurations where we varied the CPU allocated to Dom_0 relative to Dom_1, e.g., in *Conf_0.25*, Dom_0 is allocated 0.25 of the CPU allocated to Dom_1.

5.1 Impact of Different Scheduler Parameters and Dom_0 Weight

In this section, we aim to answer the following questions:

- How sensitive are I/O intensive applications to the amount of CPU allocated to Dom_0? Does allocation of a higher CPU share to Dom_0 mean a better performance for I/O intensive applications?
- How significant is the impact of scheduler parameters on application performance, e.g., context switch allowance C in BVT and period P_i in SEDF?

Figure 1(a) shows web server throughput for BVT with context allowance C set to 1 ms, 5 ms, and 50 ms. The X-axis presents the results of experiments for 5 different configurations where the CPU weights allocated to Dom_0 relative to Dom_1 are 0.25, 0.5, 1, 2, and 4.

We first note that the web server throughput is quite sensitive to Dom_0 weight for all three schedulers. Second, BVT with larger values for C supports higher web server throughput. The difference in performance is significant: when Dom_0 and Dom_1 have equal weights (Conf_1) web server throughput with context allowance $C = 1\ ms$ is 85% lower than for $C = 50\ ms$. Why? With a larger context allowance, the currently running domain is allowed to execute longer before it is preempted by another runnable domain. When Dom_0 is assigned a higher weight, it gets a higher priority when it unblocks. Intuitively, it leads to a situation where Dom_0

preempts the running guest domain on each incoming interrupt and ends up processing fewer I/O events per execution period at a higher cost (due to context switch overhead). Increasing C alleviates this problem because it lets the guest VM execute for slightly longer before being preempted by Dom_0, and as a result, Dom_0 can more efficiently process multiple I/O events accumulated over time.

Figures 1(b) and 1(c) show web server throughput for SEDF in WC and NWC-mode respectively with different granularity for periods P_i of 10 ms, 100 ms, and 1000 ms. SEDF with a smaller time period makes a fair share allocation at smaller time granularity, while with a larger time period the algorithm may result in "burstier" CPU allocation. When Dom_0 and Dom_1 have equal weights (Conf_1) SEDF scheduler in WC-mode with 10 ms period supports almost 40% higher web server throughput compared to 1000 ms period (50% throughput improvement in the NWC-mode).

To get additional system performance metrics and to gain some insight into the schedulers' behavior we analyzed monitoring results from XenMon tool [10] that reports resource usage of different domains and some scheduling information such as how often a domain has been scheduled, its average waiting time for CPU allocation (i.e., being in the run queue), etc.

Figures 2 and 3 show CPU usage by Dom_0 and Dom_1 for web server experiments performed with BVT and SEDF under different scheduler's parameters.

First, we see that Dom_0 (which performs I/O processing on behalf of the guest domains) consumes a significant share of CPU. Second, while the Dom_0 weight is varied in a significant range (from 0.5 to 4 relative to Dom_1 weight) the CPU usage by Dom_0 varies in a rather limited range between 33% to 45% for BVT and SEDF in WC-mode. Third, that the limited variation in CPU usage might lead to a drastic difference in application performance.

Since we observe the most significant difference in web server performance under BVT and SEDF schedulers with different parameters for the configuration where Dom_0 and Dom_1 have equal weights (Conf_1), we first analyze and compare resource usage by Dom_0 and Dom_1 in this configuration. The results are shown in Table 1 below.

Table 1 shows that different scheduler parameters such as decreased context switch allowance C in BVT and increased period in SEDF lead to a relatively small increase in CPU usage by Dom_0 while causing a drastic change in the application performance. In case of WC-mode, one can say that, additionally, there is also a small decrease in CPU usage by Dom_1, and this smaller CPU allocation to the application can explain worse web server performance and its lower throughput.

CPU Scheduler Type	Dom_0 Util (%)	Dom_1 Util (%)	Tput req/sec
BVT, context allow.= 50 ms	35	64	934
BVT, context allow.= 5 ms	38	61	817
BVT, context allow.= 1 ms	43	56	510
SEDF, wc, period=10 ms	35	59	696
SEDF, wc, period=100 ms	41	59	632
SEDF, wc, period=1000 ms	43	53	499
SEDF, nwc, period=10 ms	27	50	615
SEDF, nwc, period=100 ms	35	50	504
SEDF, nwc, period=1000 ms	40	50	419

Table 1: CPU usage by Dom_0 and Dom_1 and web server throughput.

However, in case of SEDF scheduler in NWC-mode, it is not true. For all the three values of period (10 ms, 100 ms, and 1000 ms) the CPU usage by Dom_1 is the same: it is at its maximum value of 50% (note that in NWC-mode, when Dom_0 and Dom_1 have equal weights they are entitled to the maximum of 50% CPU usage).

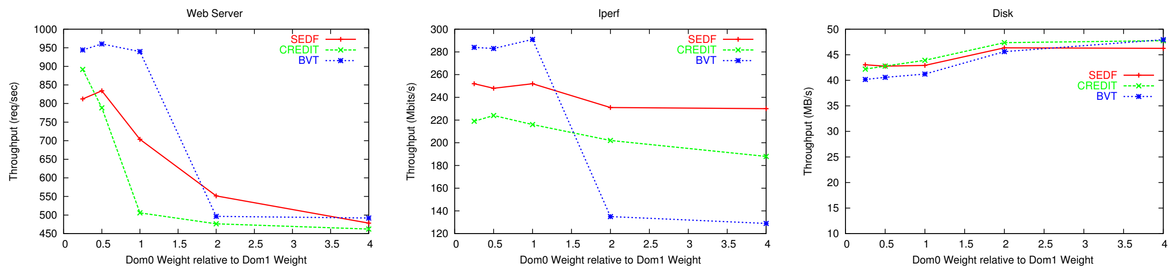
As for CPU usage by Dom_0, we observe that with larger time periods, SEDF allocates higher CPU share to Dom_0. For example, a period of 10 ms results in 27% of CPU allocation to Dom_0, and with period of 1000 ms, the CPU allocation to Dom_0 is increased to 40%, while in contrary, web server throughput, drops from 615 req/sec to 419 req/sec, causing the 33% decrease in web server throughput.

At first glance, the lower web server throughput achieved by the configuration with higher CPU share to Dom_0 seems like a contradiction. To clarify and explain this phenomenon, we analyze some additional, low level scheduling and system metrics. XenMon reports an *execution count* metric that reflects how often a domain has been scheduled on a CPU during the measurement period (e.g., 1 second). XenMon also provides *I/O count* metric that is a rough measure of I/O requested by the domain.

Table 2 below shows the number of execution periods per second and the I/O count per execution period for BVT and SEDF as discussed above and the configuration with Dom_0 and Dom_1 having equal weights (Conf_1).

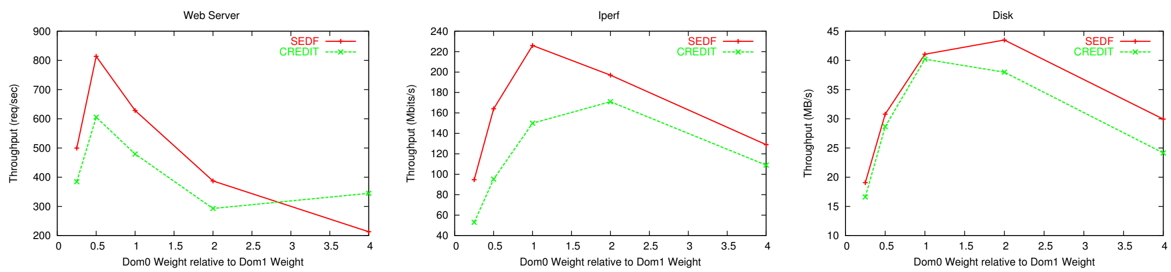
CPU Scheduler Type	ex/sec	i/o count/ex
BVT, context allow.= 50 ms	1127	27.3
BVT, context allow.= 5 ms	3080	8.6
BVT, context allow.= 1 ms	6409	2.6
SEDF, wc, period=10 ms	2478	6.9
SEDF, wc, period=100 ms	5124	3
SEDF, wc, period=1000 ms	6859	1.9
SEDF, nwc, period=10 ms	451	34.6
SEDF, nwc, period=100 ms	4635	2.8
SEDF, nwc, period=1000 ms	7292	1.5

Table 2: The number of execution periods per second of Dom_0 and I/O count per execution period in Dom_0.



(a) Web server (b) iperf (c) Disk

Figure 4: Evaluating the three schedulers (WC-mode) for different workloads.



(a) Web server (b) iperf (c) Disk

Figure 5: Evaluating the three schedulers (NWC-mode) for different workloads.

Indeed, in case of a smaller context switch allowance C in BVT or a larger time period in SEDF, both schedulers exhibit a similar behavior: they schedule Dom_0 much more often (see the increased number of execution periods). However, frequently scheduled Dom_0 is processing fewer I/O events that are accumulated in between the Dom_0 execution periods. This type of scheduling leads to a higher context switch overhead and to a worse web server performance. In such a way, while the observed CPU usage by Dom_0 is higher, in fact, it performs less useful work which manifests itself as degraded application performance.

Figures 4 and 5 show the performance of the three workloads for the three schedulers in the WC and NWC modes respectively. For brevity, we omit detailed analysis of these experiments and summarize below:

- I/O intensive applications are highly sensitive to the amount of CPU allocated to Dom_0. The problem of adequate CPU allocation to Dom_0 and efficient CPU scheduling becomes even harder when multiple VMs with diverse set of applications are competing for I/O processing in Dom_0;
- Application performance varies significantly under different schedulers even when the schedulers are configured with the same CPU allocation shares;
- Application performance is significantly worse under NWC-mode when compared to WC-mode (when using similar shares). NWC-mode is an operational requirement for workload managers (it is used to support performance isolation and to deliver resource guarantees between applications). Optimizing CPU schedulers to support a more efficient CPU allocation under NWC-mode is an often overlooked problem.

Thus the choice of the CPU scheduler and its configuration can significantly impact application performance despite supporting similar resource allocation models. In an environment where different servers may potentially run different CPU schedulers with varying configurations, the job of the workload manager becomes even more complex: migrating a VM to a different node with more resources does not necessarily result in better application performance. Hence, one interesting open question is whether virtualization environments must employ a single CPU scheduler with fixed parameters to successfully manage heterogeneous workloads.

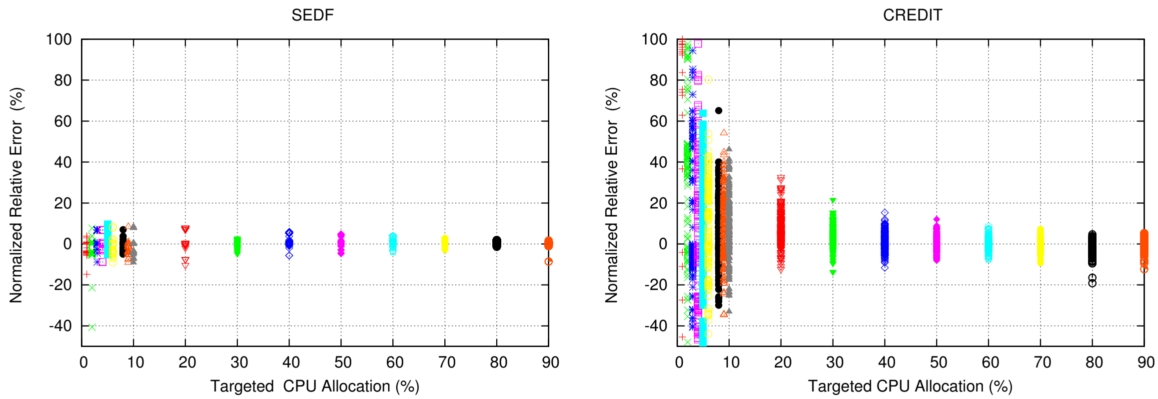
5.2 Scheduler CPU Allocation Accuracy

A traditional metric used in scheduler's analysis and comparison is the *error of CPU allocation*.

This metric is also important in practice as we discussed in Section 3. Since the decision of workload manager's controller is based on a difference between assigned and consumed CPU allocation, a scheduler with significant error in CPU allocation may cause unstable controller behavior and as a corollary, lead to a poor application performance.

To evaluate the CPU allocation error in NWC-mode for SEDF and Credit schedulers, we designed a simple benchmark, called ALERT (**AL**location **ER**ror **T**est):

- Dom_0 is allocated a fixed, 6% CPU share during



(a) SEDF (b) Credit

Figure 6: Benchmarking with ALERT: CPU allocation error under SEDF versus Credit schedulers in Xen.

the all benchmark experiments; it is more than sufficient to run XenMon monitoring tool;

- the guest domain Dom_1 executes a cpu-hungry loop;
- for each benchmark point i the CPU allocation to Dom_1 is fixed to A_i, and each experiment i continues for 3 min;
- the experiments are performed with A_i = 1%, 2%, 3%, ..., 10%, 20%, ... , 90%.

Note that under this benchmark there are no any contention for resources, i.e., there are always enough CPU resources for Dom_1 to receive its CPU share. ALERT is truly the simplest test to verify the accuracy of a scheduling algorithm. While it does not guarantee the same CPU allocation accuracy when there are competing VM's, one can easily extend ALERT to test the CPU allocation error for multiple VMs, as well as for a case with multiple VCPUs per VM.

Let U_i^k denote CPU usage of Dom_1 measured during the k-th time interval in benchmark experiment i, e.g., we sample CPU usage of Dom_1 at the second time scale in the ALERT experiments.

If a CPU scheduler works accurately we should see that for $X\%$ of CPU allocation to Dom_1 it should consume X % of CPU, i.e., ideally, $U_i^k = A_i$ for any k-th time interval in benchmark experiment i.

Let Er_i^k denote *a normalized relative error of CPU allocation* defined as follows:

$$Er_i^k = (A_i - U_i^k)/A_i$$

We execute ALERT benchmark under SEDF and Credit schedulers in Xen. The Credit scheduler uses 30 ms as a time slice for CPU allocation as described in Section 4. To match the CPU allocation time granularity we use 10 ms period in SEDF in our comparison experiments.

Figures 6 a) and b) show the normalized relative errors of CPU allocation with ALERT for SEDF and Credit schedulers respectively at 1 second time granularity, i.e., we compare the CPU usage U_i^k of Dom_1 measured at each second during experiment i with the assigned CPU allocation value A_i. X-axes represent the targeted CPU allocation, Y-axes show the normalized relative error.

Each experiment is represented by 180 measurements (3 min = 180 sec). Thus, each "bar" in Figures 6 a) and b) has 180 points and the bar's density somewhat reflects the error distribution. As Figure 6 a) shows the CPU allocation errors under SEDF are consistent and relatively small across all of the tested CPU allocation values. The Credit scheduler has overall much higher allocation error as shown in Figure 6 b). The errors are especially high for smaller CPU allocation targets, i.e., below 30%. [1]

Figure 7 presents the distribution of all the errors measured during the ALERT's experiments for SEDF and Credit relatively. We plot the normalized relative errors measured at 1 second time scale for all the performed experiments in ALERT ($18 \times 180 = 3240$ data points). It is a special version of the CDF (*cumulative distribution function*), where we plot the CDF of the negative errors (with errors ordered in decreasing order) normalized with respect to all the errors as well as the complementary CDF of positive errors (with errors ordered in increasing order). We call it CDF_-^+.

Figure 7 presents CDF_-^+ of both positive and negative errors with respect to all the errors. This way, we can see that the Credit scheduler is over-allocating the CPU share more often than under-allocating one, while for SEDF the under-allocating is a more typical error. As apparent from Figure 7 the Credit scheduler has a much higher CPU allocation error compared to SEDF scheduler:

- for Credit, 10% of the points have the negative errors worse than -9.9% while for SEDF only 0.03% of points have the error in this range;

[1] We had to limit the shown error in Figure 6 b) to the range of $[-50\%, 100\%]$ for visibility: the actual range of the observed errors is $[-100\%, 370\%]$.

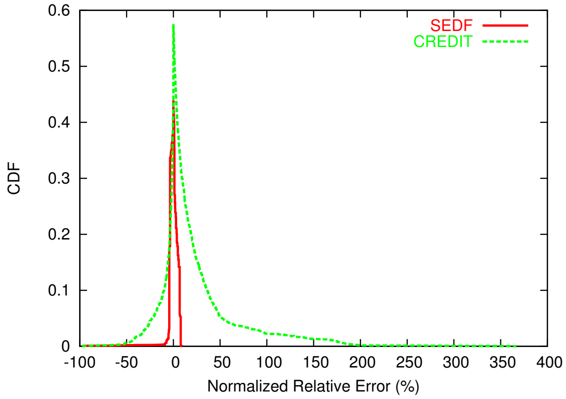


Figure 7: CDF_{-}^{+} of CPU allocation errors.

- for Credit, 10% of the points have the positive errors larger than 36.3% while for SEDF there are no error in this range: the maximum positive error is 9.52%.

Figure 8 shows the normalized relative errors of CPU allocation at 3 min time scale, i.e., we compare the targeted CPU allocation with average CPU utilization measured at the end of each ALERT experiment (each experiment runs for 3 min).

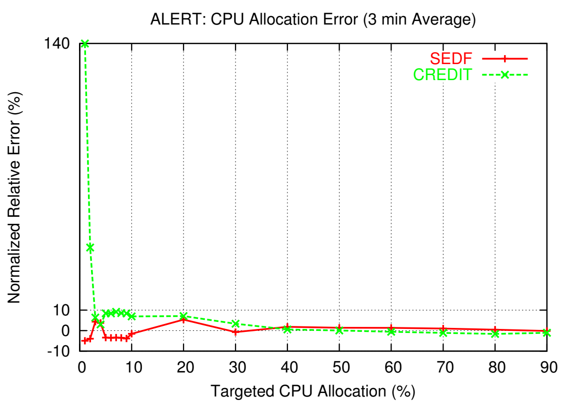


Figure 8: Normalized relative error at a longer time scale of 3 min.

Overall, SEDF and Credit show comparable CPU allocation averages over longer time scale. However, the Credit's errors are still significantly higher than SEDF's errors for CPU allocation in the range [1%, 30%] as shown in Figure 8.

Since many advanced management tools like workload manager controllers (see Section 3) rely and depend on accurate CPU allocation at a fine time granularity, it is important to optimize the CPU scheduler behavior and minimize the allocation error as well as to augment the provided CPU schedulers with measured allocation error results.

6 Discussion

As VM technologies evolve, their I/O model will certainly undergo some changes. However, it is unlikely that resource allocation problems (such as those described in this paper) will disappear anytime soon. For instance, for fully virtualized guests, Xen is considering moving the I/O emulation out of Dom_0 and into per-VM "stub domains" [12]. In this case, the problem becomes even more challenging because resources must now be cleverly allocated across twice the number of schedulable entities. Similarly, virtualization-aware I/O devices (with multiplexing capabilities in hardware) will ease the problem slightly, however the CPU might still remain on the critical path for most I/O operations. It is conceivable, however, that a combination of better I/O hardware and multi-core processors will alleviate the problem to some extent in the case where the number of available cores exceeds the required level of inherent parallelism in the workload.

Thus far all our experiments have focused on one particular virtualization platform — Xen. Our motivation for using Xen, among others, was the availability of its source code and the freedom to modify it. However, we stress that *as long as the I/O model involves services being provided by an entity external to the VM*, resource allocation problems *will* remain. The only difference is that in Xen's case (where we have a split device driver model and a choice of CPU schedulers), the problems are exposed to the end user and in platforms where the device drivers are in the hypervisor and there is no choice in CPU schedulers, the end user is not directly concerned with these issues. However, the developers of the hypervisor will most certainly need to address the problem of allocating resources to the hypervisor for providing services while making decisions on the choice of scheduler and scheduler parameters.

At first glance, it may seem that the choice of VM scheduler and parameter configuration is not relevant to most users because they are often shielded from such decisions. However, our experience suggests that "reasonable defaults" are not very useful beyond toy experiments. For any serious VM deployment, the platform will need to give users control over the scheduling parameters and provide flexible mechanisms that allow a wide variety of resource allocation policies. Our experiences with Xen's CPU schedulers suggests that our understanding of VM resource allocation issues is far from complete, and opens several interesting avenues for future research.

7 Aknowledgements

The authors owe warm thanks to Stephan Diestelhorst and Timothy Roscoe for discussions on implementation issues and algorithm details of SEDF, to Emmanuel Ackaouy for explanations of Credit scheduler and his hearty reaction to the first informal "accuracy" results that led to a significantly improved Credit scheduler's implementation that was presented in this paper.

We also thank our HP colleagues John Sontag, Tom Christian, Scott Rhine, Thomas Turicchi, Jonathan Ross, Cliff McCarthy, Bill Blanding, Aland Adams, Al Stone, and Joseph Szczypek for related explanations of HP-UX, gWLM, and HPVM scheduling issues, comments, and useful feedback on this work.

References

[1] Credit Scheduler. `http://wiki.xensource.com/xenwiki/CreditScheduler`.

[2] HP-UX Workload Manager. `http://www.hp.com/products1/unix/operating/wlm/`.

[3] Httperf. `http://www.hpl.hp.com/research/linux/httperf/`. Last accessed 1/17/2006.

[4] IBM Enterprise Workload Manager. `http://www.ibm.com/developerworks/autonomic/ewlm/`.

[5] Iperf: The TCP/UDP Bandwidth Measurement Tool. `http://dast.nlanr.net/Projects/Iperf`.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM SOSP*, 2003.

[7] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *Proceedings of the 17th ACM SOSP*, 1999.

[8] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Reconstructing I/O. Technical report, 2004.

[9] T. Garfinkel and M. Rosenblum. When virtual is harder than real: Security challenges in virtual machine based computing environments. In *Proceedings of the 10th HotOS*, 2005.

[10] D. Gupta, R. Gardner, and L. Cherkasova. XenMon: QoS Monitoring and Performance Profiling Tool. Technical report, HPL-2005-187, 2005.

[11] I. M. Leslie, D. Mcauley, R. Black, T. Roscoe, P. T. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal of Selected Areas in Communications*, 1996.

[12] I. Pratt. Xen Roadmap. `http://wiki.xensource.com/xenwiki/XenRoadMap`.