Xen and Co.: Communication-aware CPU Scheduling for Consolidated Xen-based Hosting Platforms

Sriram Govindan Arjun R. Nath Amitayu Das
Bhuvan Urgaonkar Anand Sivasubramaniam

{sgovinda, anath, adas, bhuvan, anand}@cse.psu.edu
The Pennsylvania State University, University Park, PA, 16802.

Abstract

Recent advances in software and architectural support for server virtualization have created interest in using this technology in the design of consolidated hosting platforms. Since virtualization enables easier and faster application migration as well as secure co-location of antagonistic applications, higher degrees of server consolidation are likely to result in such virtualization-based hosting platforms (VHPs). We identify a key shortcoming in existing virtual machine monitors (VMMs) that proves to be an obstacle in operating hosting platforms, such as Internet data centers, under conditions of such high consolidation: CPU schedulers that are agnostic to the communication behavior of modern, multi-tier applications. We develop a new communication-aware CPU scheduling algorithm to alleviate this problem. We implement our algorithm in the Xen VMM and build a prototype VHP on a cluster of servers. Our experimental evaluation with realistic Internet server applications and benchmarks demonstrates the performance/cost benefits and the wide applicability of our algorithms. For example, the TPC-W benchmark exhibited improvements in average response times of up to 35% for a variety of consolidation scenarios. A streaming media server hosted on our prototype VHP was able to satisfactorily service up to 3.5 *times as many clients as one running on the default Xen.*

Categories and Subject Descriptors D.4.1 [*Process Management*]: Scheduling; D.4.4 [*Communications Management*]: Network communication; D.4.8 [*Performance*]: Measurements

General Terms Algorithms, Design, Experimentation, Performance

Keywords Virtual machine monitor, Xen, multi-tier application, CPU scheduler

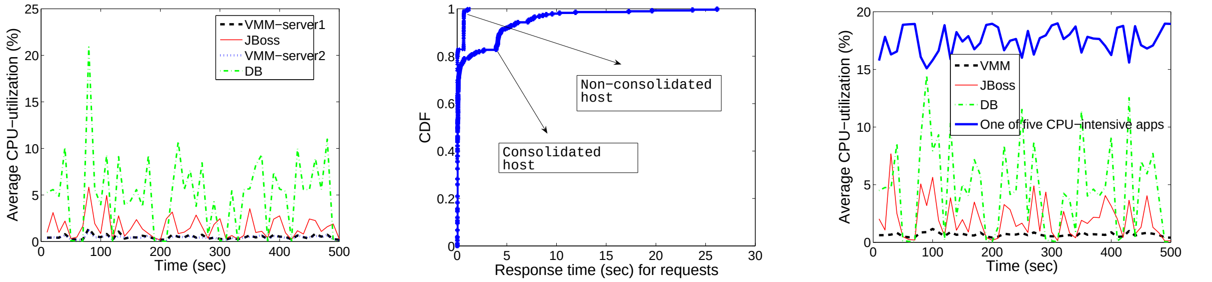
1. Introduction and Motivation

The recently resurgent research in server virtualization has fueled interest in using this technology to design consolidated hosting platforms. In this emerging hosting model, each physical server in the cluster runs a software layer called the Virtual Machine Monitor (VMM) that virtualizes the resources of the server and supports the execution of multiple Virtual Machines (VMs). Each VM runs a separate operating system within it and the VMM provides safety and isolation to the overlying operating systems. The development of highly efficient VMMs [43, 6, 40] as well as the evolution of architectural support for them [19] is helping reduce the overheads associated with virtualization. As a result, these overheads may be far outweighed by the benefits offered by VMMs such as the ease of application migration and secure co-location of un-trusting services [30, 27, 11].

Cluster-based hosting platforms have received extensive attention in several research communities such as those dealing with operating systems [5, 39, 31], parallel/distributed computing [4, 35, 26, 36, 46], and scheduling theory [1]. With the burgeoning of various kinds of Internet server applications that cater to domains such as e-commerce, education, and entertainment, recent research efforts have focused on the design of Internet data centers that host and manage them in return for revenue. These applications are typically communication and disk-I/O intensive, adhere to highly modular software architectures with multiple communicating *tiers*, [1] and require resource guarantees from the hosting platform to provide satisfactory performance to clients who access them over the Internet. The use of virtualization for cost reduction and easier management is being actively explored in such Internet data centers as well

[1] Note that the term *tier* is generally used to collectively denote multiple functionally identical components of an application. For example, the Web tier in an e-commerce application may consist of multiple replicated Web servers. We do not make such a distinction in this work and our techniques apply equally well to applications with multi-component tiers.

VEE'07, June 13–15, 2007, San Diego, California, USA.
Copyright © 2007 ACM 978-1-59593-630-1/07/0006. . . $5.00



(a) CPU usage: isolated servers (b) Client response times (c) CPU usage: consolidated server

Figure 1. Performance degradation of a communication-intensive application placed on a consolidated server despite allocating sufficient resources.

as in those used internally by organizations to consolidate the IT infrastructure of their various departments. The design of such Virtualization-based Hosting Platforms (VHPs) to host modern applications raises some novel design considerations. This paper presents the design and evaluation of mechanisms to address such issues in the Xen VMM. We choose *Xen* because it is open source and is increasingly popular among data center providers [2, 44].

The need for communication-aware CPU scheduling: Server virtualization opens up the possibility of achieving *higher server consolidation* and *more agile dynamic resource provisioning* than is possible in traditional platforms. Ensuring that the applications experience satisfactory performance even under such consolidation requires the hosting platform to perform (i) *resource requirement estimation* for the hosted applications, done either by application profiling [32, 39] or using analytical models [13, 7, 38] and (ii) *application placement* that involves ensuring that the requirements of co-located application tiers do not exceed the capacity of the server used to host them, usually based on a simple aggregation of resource requirements (deterministic or statistical) [10, 39, 31]. While these approaches have been shown to work satisfactorily under low or moderate server utilization, they may not suffice in conditions of high resource utilization that will accompany the high degrees of consolidation likely in VHPs. Specifically, over and beyond ensuring that we provide each application tier with its CPU needs, an equally important consideration is *when* this CPU capacity is provided to it.

To illustrate this, we depict the performance experienced by an implementation of the TPC-W benchmark [34] from New York University consisting of two tiers - a JBoss tier that implements the application logic and interacts with the clients and a Mysql-based database tier that stores information about items for sale and client information - under conditions of high consolidation. Figure 1(a) presents the CPU usage of the two tiers when the application was run with each tier on a separate dedicated physical server running the Xen VMM [6]. We then ran this application with all its tiers consolidated on a single server running Xen along with 5 CPU-intensive applications, while ensuring that the two tiers received the same resource allocations. We used a reservation-based scheduler in Xen to achieve the same CPU allocation as in Figure 1(a) - Figure 1(c) confirms this. Same memory was ensured by statically providing each tier of TPC-W the same virtual RAM size. Finally, the network and disk bandwidths received were the same in both cases, since the 5 new applications did not perform any I/O activity. Figure 1(b) compares the performance experienced by the clients of this application under the two scenarios. We find a significant degradation in the response time of TPC-W under the consolidated scenario. *Why did the performance degrade despite providing the same resource allocations?* The reason for this is that for applications with communicating components, providing enough CPU alone is not enough - an equally important consideration is to provide CPU at the *right time*. Due to the presence of the 5 CPU-intensive applications on the consolidated server, the TPC-W tiers spend large amounts of time waiting for a chance to communicate, resulting in degraded response times. Since these delays depend on the order in which the CPU scheduler chooses competing co-located application tiers, we call such delays as *scheduling-induced delays*. We realize that similar delays could occur in traditional operating systems. However, virtualization enables secure co-location of applications written for heterogeneous operating systems, something that was more difficult to realize in traditional hosting platforms. Hence, virtualized environments are more likely to experience higher degrees of consolidation, and consequently are more likely to face this problem.

Problem: *Can a server in a VHP schedule hosted VMs in a communication-aware manner to enable satisfactory application performance even under conditions of high consolidation, while still adhering to the high-level resource provisioning goals in a fair manner?*

1.1 Research contributions

We develop a CPU scheduling algorithm for a VMM that incorporates the I/O behavior of the overlying VMs into its decision-making. The key idea behind our algorithm is to introduce short-term unfairness in CPU allocations by preferentially scheduling communication-oriented applications over their CPU-intensive counterparts. Our algorithm works solely based on communication events local to the server. Furthermore, it maintains an administrator-specified upper bound on the time-granularity over which deviations from fair CPU allocations are allowed.

We identify efficient ways of implementing this algorithm in the state-of-the-art Xen VMM. We use the Xen VMM, enhanced with our algorithm, to build a prototype VHP of a collection of physical servers. We explore the pros and cons of our implementation by experimenting with realistic and representative Internet sever applications/benchmarks. Our evaluation demonstrates the benefits of our approach in improving the management of highly consolidated hosting platforms. For example, the TPC-W benchmark exhibited improvements in average response times of up to 35% for a variety of consolidation scenarios. A streaming media server hosted on our prototype VHP was able to satisfactorily service up to 3.5 times as many clients as one running on the default Xen.

1.2 Outline

The rest of this paper is organized as follows. We present background material on Server virtualization, the Xen VMM and VHPs in Section 2. We discuss the design and implementation of our communication-aware CPU scheduling in Section 3. We describe our experimental setup and evaluation in Section 4. We present related work in Section 5. Finally, we present concluding remarks in Section 6.

2. Background and System Overview

In this section, we provide an overview of Server virtualization, the Xen VMM and VHPs.

2.1 Server Virtualization

Virtualization refers to the creation of a virtual (rather than actual) version of a resource/entity, such as an operating system, a server, a storage device, network resources, etc. In our research, we use this term to denote the *virtualization of a server at the operating system level*. This is achieved by a software layer called the Virtual Machine Monitor (VMM) that runs directly on server hardware. A VMM virtualizes the resources of a physical server and supports the execution of multiple virtual machines (VMs) [15, 6, 33, 43]. Each VM runs a separate operating system within it and the VMM provides safety and isolation to the overlying operating systems. The VMM manages the sharing of CPU, memory, and I/O devices among the VMs. Each VM is provided a set of virtual I/O devices for which its operating system implements drivers. VMMs have been a topic of extensive research for over four decades due to their numerous uses including secure co-location of operating systems or applications, facilitating migration, enabling the existence of legacy applications on newer platforms, etc [15, 30, 33, 6, 11, 27, 29].

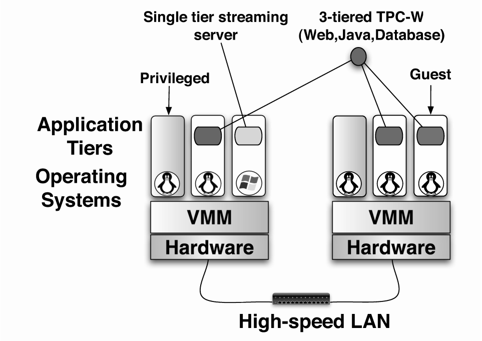


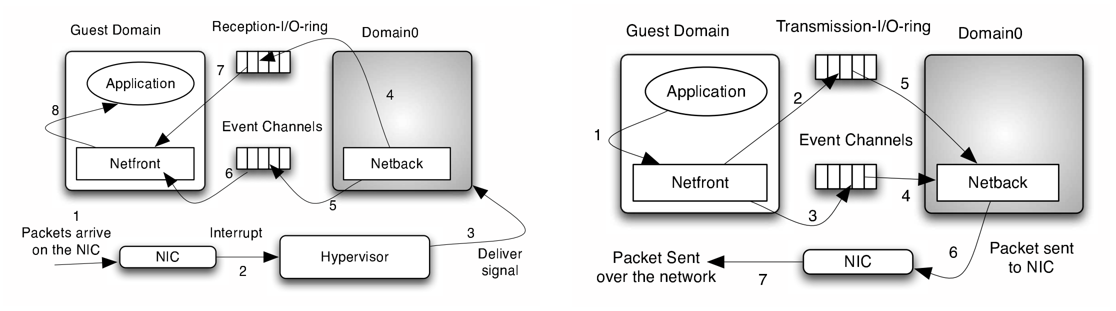
Figure 2. Illustration of hosting in a Xen-based VHP.

2.2 The Xen VMM

We use the Xen VMM in our research [6] and conduct the remaining discussion in its context. Figure 2 shows two VMMs supporting three VMs (called *guest domains or simply Domains* in Xen) each. One of the VMs, called *Domain0*, implements the real device drivers, communicates with the actual hardware and does the translation between virtual and real I/O activity.

Network I/O virtualization in Xen: Each guest domain implements a driver for its virtual NIC that is called its *netfront* driver. *Domain0* implements a *netback* driver which acts as an intermediary between netfront drivers and the device driver for the physical NIC. The device driver, also part of *Domain0*, can access the physical NIC but interrupts from the NIC are first handled by the hypervisor which in turn sends virtual interrupts via *event-channels* to *Domain0*. Event-channels are an asynchronous notification mechanism used for communication between domains. While these event-channels are strictly for notification, Xen uses a shared-page mechanism called *network-I/O-rings* (one each for reception and transmission per domain) for inter-domain message passing. To enable fast I/O, Xen employs a zero-copy, page-flipping mechanism to exchange pages of data between the guests' netfront drivers and *Domain0*'s netback driver.

Now we describe the key steps in network transmission and reception in the context of Xen. When a network packet arrives at the physical NIC for any domain, an interrupt is delivered to the hypervisor which in turn notifies *Domain0* of packet arrival as described above. Subsequently, when *Domain0* is scheduled, netback checks the destination of the packets that have arrived. *Domain0* notifies the recipient guest domains and updates the reception-I/O-rings to copy the packets into their address spaces. When the target guest domain is scheduled next, it sees packets that have ar-



(a) Reception

(b) Transmission

Figure 3. Network I/O virtualization in Xen.

rived for it and processes them as any standard OS would do. Similarly, when packets are sent by a guest domain, it notifies *Domain0* of the packets to be transmitted, again via its event-channel. Upon its next scheduling, *Domain0* delivers the packets to the NIC. Figure 3 presents these steps in detail.

2.3 A virtualized hosting platform

Our hosting model assumes a large cluster of high-end servers (with dual processors and a few GB of memory) interconnected by a high bandwidth network for communication. In addition, many of these servers are also connected to a consolidated high capacity storage device/utility through a Storage Area Network which facilitates data sharing and migration of applications between servers without explicit movement of data. These servers are connected via some gateway to the Internet to service end-user requests from clients of the application service providers.

Each server in our VHP runs a Xen hypervisor on top of the native hardware. Each application tier and its associated OS run within a Xen guest domain. Figure 2 provides an illustrative example. Our enhanced hypervisor implements a modified CPU scheduler to achieve communication-aware scheduling of domains.

3. Communication-aware Scheduling in Xen

We assume that the VMM provides a CPU scheduler that allows applications to specify guarantees on the CPU allocations that they desire for their tiers from the platform. Several such schedulers exist, most notably proportional-share schedulers and reservation-based schedulers [14, 42, 9, 21, 28]. The problem of determining the CPU allocations appropriate for the performance needs of an application is orthogonal to this research. We point the reader to extensive existing work in this area [39, 31]. The Xen hypervisor implements an algorithm called *Simple Earliest-Deadline-First* (SEDF) that allows domains to specify lower bounds on the CPU reservations that they desire [2]. Specifically, each domain specifies a pair (*slice, period*) asking for *slice* units of the CPU every *period* time units. The hypervisor ensures that the specified reservation can be provided to a newly created domain (or a domain that desires to change its CPU reservation). We assume that a domain is not admitted if its reservation cannot be satisfied. The residual CPU capacity is shared among the contending domains (including *Domain0*) in a round-robin fashion. We now develop a CPU scheduling algorithm that incorporates the communication activities of the hosted domains into its decision-making. We build our algorithm *on top of* SEDF in the sense of retaining SEDF's basic feature of guaranteeing the specified *slice* to a domain over every *period*. Our algorithm attempts to preferentially schedule communication-sensitive domains over others while ensuring that the resulting unfairness in CPU allocations is bounded; the latter is ensured by exploiting the guarantees offered by SEDF.

The Xen hypervisor also implements a proportional share scheduler called *Borrowed Virtual Time* (BVT) [14] scheduler. This scheduler has a set of parameters which can be configured to provide low latency dispatch for the I/O intensive domains. However, the effectiveness of this approach significantly depends on the careful selection of these parameters. Furthermore, these parameters may also need to be dynamically adjusted to suit the varying application I/O demands. These are non-trivial activities and may require frequent administrator intervention. On the other hand, our scheduler tunes itself to the applications' current I/O intensities.

The Latest version of Xen includes a proportional share scheduler called the *Credit-based* [12] Scheduler. It does better load balancing on multi-processor systems than earlier schedulers and also provides mechanisms for low latency dispatch for I/O intensive domains. The *Credit scheduler* was not available during the time of our implementation. In future, we intend to analyze how to incorporate our

[2] We use the term *domain* to denote a guest domain as well as the *tier* hosted within it henceforth; we will use the term tier only when a distinction is necessary.

communication-aware scheduling technique in this scheduler as well.

We begin our discussion by defining the goal of our scheduler and identifying ways in which it might achieve it. Following this, we describe various components of our scheduler in detail along with the considerations that arise for their implementation in the Xen hypervisor.

3.1 Classifying scheduling-induced delays

In a consolidated server, a domain can experience scheduling-induced delays (as was discussed in Section 1) in its communication activities due to the CPU contention with other co-located domains, including *Domain0*. The goal of our CPU scheduler is to *reduce the aggregate scheduling-induced delay for the hosted domains while still providing guarantees on CPU allocations.*

With the background presented in Section 2.2 and Figure 3, we identify three sources of scheduling-induced delays.

1. **Delay associated with the scheduling of *Domain0*:** This is either (i) the duration between a packet reception at the physical NIC and when *Domain0* is scheduled next to set up an event channel notification for the recipient guest domain or (ii) the duration between when a transmitting domain copies a packet into the transmission-I/O-ring of *Domain0* and when *Domain0* gets scheduled next to actually send it over the physical NIC. These delays can be reduced by (i) scheduling *Domain0* soon after a packet is received by the physical NIC and (ii) soon after a domain does a send operation over its virtual network interface, respectively.
2. **Delay at the recipient:** This is the duration between when *Domain0* sets up an event channel notification for the recipient domain (on packet arrival) and when the recipient domain gets scheduled next to receive the packet. This delay can be reduced by scheduling the recipient domain soon after the reception of a packet for it in *Domain0*.
3. **Delay at the sender:** This is the extra delay, before a domain sends a network packet (on its virtual NIC), induced by the hypervisor scheduling other domains in between, compared to running this domain in isolation. Notice that unlike reception, sending a packet is an event that can only be anticipated. This delay can be reduced by anticipating when a domain would be ready to send a packet and scheduling it close to that time.

With this intuition, we now consider ways to reduce each of these three types of delay in detail. Figure 4 presents examples of these delays for a two-tiered TPCW application with a Java tier and a database tier hosted on a separate physical machine. Here, delays $d1, d2, d4$, and $d5$ are of type 1, while delays $d3$ and $d6$ are of type2. There are no type3 delays in this example.

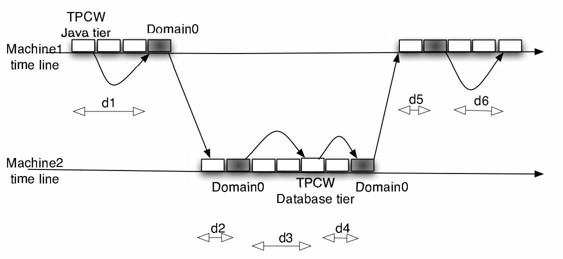


Figure 4. Sources of scheduling-induced delays.

3.2 Preferential scheduling of recipient

Scheduling a domain close to the reception of a packet is easy to achieve in theory, since this is purely reactive. However, we would like to devise a general approach that can choose between multiple recipient domains. *Which domain should be chosen out of multiple recipients?* Our scheduler implements a naturally appealing heuristic that picks the domain that is likely to experience the most overall reduction in scheduling-induced delay, that is, the domain that has received the most number of packets.

It should be pointed out that our approach does not cause a domain receiving high-intensity traffic to starve other domains. Since we ensure that our algorithm continues to provide the reservations guaranteed by the default SEDF, such a domain will only be preferentially chosen so long as it has received less than its *slice* for the ongoing *period*. The expected outcome of this approach is to delay the scheduling of non-recipient domains in favor of the recipients. The resulting unfairness in CPU allocations is limited to durations smaller than a *period*.

Implementation considerations: In Xen, each domain, including *Domain0*, is given a page that it shares with the hypervisor. We use these pages to maintain various I/O related statistics needed by our scheduler and call these *book-keeping pages*. For keeping track of the number of packets received and waiting within *Domain0*, we introduce *network-reception-intensity* variables, one for each domain, stored in the book-keeping page of *Domain0*. These variables are initialized to zero upon domain start up by the hypervisor. Subsequently, these variables are updated as follows. Whenever *Domain0* runs (we will describe when this happens momentarily), the netback driver figures out which domains have received packets since the last time *Domain0* was de-scheduled and uses the page-flip mechanism to copy pages containing them to the appropriate domains. It then uses the number of pages flipped with each domain as an indicator of the number of packets [3] received by that domain and increments the *network-reception-intensity* variable of the corresponding recipient domain in the book-keeping

[3] This works accurately for default Xen where each packet (Ethernet frame) gets an entire page [6] regardless of its size. This becomes an estimate in some optimized versions of Xen [24].

page of *Domain0*. Whenever a recipient domain runs next, its netfront driver processes some (or all) of the packets received by it (as described in Section 2), maintaining a count of the number of packets processed in its own book-keeping page. Finally, when this domain is de-scheduled, the hypervisor reads this count and decrements the *network-reception-intensity* variable for the domain to reflect its pending value (this is in the book-keeping page of *Domain0*). See Figure 5 for a concise illustration of this.

3.3 Anticipatory scheduling of sender

As noted earlier, reducing the delay at a sender domain requires us to *anticipate* when this domain would have data to send next. Consequently, the efficacy of this approach is intimately dependent on how well the scheduler can predict such an event for a domain. While it is certainly tempting to try sophisticated prediction techniques, we take a simple low-overhead approach in this paper. We use a simple *last-value-like* prediction in which the number of packets sent by a domain during its last transmission is used as a predictor of the number of packets that it would transmit the next time it is ready to send. We use the duration Δ_{tx} between the last two transmission operations (note that any such event may involve the transmission of multiple packets) by a domain (call these time instants T_{tx} and T_{tx-1} respectively, tx is the number of transmission events since some milestone, such as a domain start up) as a predictor of the duration over which the domain is likely to indulge in a transmission again (i.e., $[T_{tx}, T_{tx} + \Delta_{tx}]$). Similar to our approach for reducing the scheduling-induced delay at a sender, when multiple domains are anticipated to transmit, we could choose to schedule the one that is expected to transmit the most packets. Also, the fairness embedded in our algorithm will limit the negative impact of any short-term unfairness caused by anticipatory scheduling of sender domains to durations less than a *period*.

Implementation considerations: We introduce an additional variable called *actual-network-transmit-intensity* in the book-keeping page of each guest domain. These variables are initialized to zero at domain start up and updated as follows. When a guest domain transmits a network packet, the netfront driver of the domain copies it to its transmit-I/O-ring, which is shares with *Domain0*, and increments the *actual-network-transmit-intensity* in its book-keeping page by one. Additionally, we introduce a *anticipated-network-transmit-intensity* variable for each guest domain in the book-keeping page of *Domain0*, also initialized to zero at domain start up. Whenever a guest domain is de-scheduled, the hypervisor adds the value of its *actual-network-transmit-intensity* variable to the corresponding *anticipated-network-transmit-intensity* variable in *Domain0*'s book-keeping page.

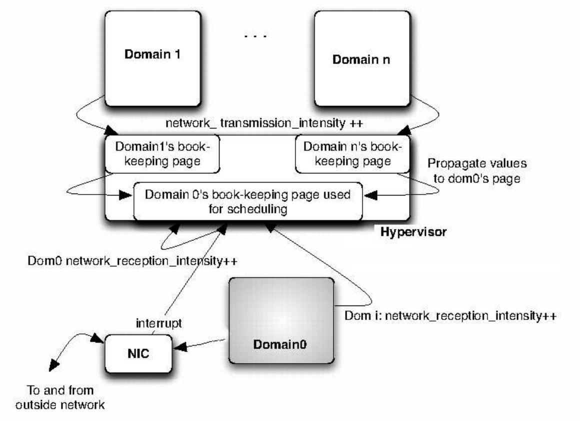

Domain 1
. . .
Domain n
network_ transmission_intensity ++
Domain1's book-keeping page
Domain n's book-keeping page
Propagate values to dom0's page
Domain 0's book-keeping page used for scheduling
Hypervisor
Dom0 network_reception_intensity++
Dom i: network_reception_intensity++
interrupt
NIC
Domain0
To and from outside network

Figure 5. Implementation of our scheduler.

3.4 Scheduling of *Domain0*

As depicted in Figures 3(a),(b) and Figure 4, *Domain0* has a crucial role in ensuring the timely delivery of received packets to domains as well as transmitting the packets sent by them (over their virtual network interfaces) on the physical interface. By default, the Xen scheduler employs a high reservation of $(15$ msec, 20 msec$)$ for *Domain0* to ensure its prompt scheduling. Additionally, we would like to preferentially schedule *Domain0* at times when it is likely to be on the critical path as far as our goal of minimizing scheduling-induced delays is concerned. We extend our basic approach of scheduling the domain likely to reduce the delays for most packets, to include *Domain0* as well. To achieve this, we identify two kinds of packets that would be processed when *Domain0* gets scheduled: (i) packets written by guest domains to their virtual NICs and (ii) packets received for delivery to domains and waiting in their reception-I/O-ring within *Domain0*. Note that only for *Domain0*, transmission event is deterministic whereas for the guest VMs, it could only be anticipated.

Implementation considerations: When a packet arrives at the NIC card for any domain, an interrupt is delivered to the hypervisor which increments the *network-reception-intensity* variable of *Domain0* by 1 in the book-keeping page of *Domain0*. Notice the difference in how the *network-reception-intensity* for *Domain0* is incremented compared to those for the guest domains. Additionally, whenever a guest domain is de-scheduled, the hypervisor increments the *network-transmission-intensity* for *Domain0* by that of this domain (we already described how this quantity is updated). It should be clear by now that this update occurs in the book-keeping page of *Domain0*. As with other variables, *network-reception-intensity* and *network-transmission-intensity* for *Domain0* are initialized to zero when it starts.

Having explained these book-keeping activities, we now describe how the scheduler uses them in its decision-making. Whenever the scheduler is invoked, it simply examines the book-keeping page of *Domain0*. Notice how all the I/O statistics needed by our scheduler, maintained in various book-keeping pages, eventually get propagated to this page due to the mechanisms described above. Our scheduler picks the domain with the highest *network intensity*, which is the sum of: (i) *network-reception-intensity* and *anticipated-network-transmission-intensity* for guest domains and (ii) *network-reception-intensity* and *network-transmission-intensity* for *Domain0*. Figure 5 presents an example to help understand the overall implementation of the scheduler. Note that, after scheduling a domain based on its book-keeping variables, these variables are later adjusted to reflect the completed network activities. This ensures that we do not accumulate history and our scheduling is based only on recent network activities.

We are now ready to answer the general question that our scheduler must address, namely, *which domain among possibly multiple runnable domains - Domain0, recipient domains, and (anticipated) sender domains - should be scheduled?* We propose a "greedy" approach which picks the domain $\mathcal{D}$ that satisfies the following two conditions.

- *Respect Reservations:* Scheduling $\mathcal{D}$ would not violate the CPU reservations of any of the domains.
- *Minimize Delays:* Scheduling $\mathcal{D}$ will help reduce the scheduling-induced delay for the *most* packets. This is the greedy aspect of our algorithm that was mentioned above.

3.5 Salient features and alternatives

We now present some salient features of our algorithm and discuss some alternate design choices.

Co-ordinated scheduling and other benefits: The goal of our algorithm is reminiscent of that of the gang scheduling and co-scheduling algorithms developed in the parallel/distributed systems literature. In particular, like implicit co-scheduling algorithms, it is expected to achieve co-ordinated scheduling of various communicating tiers of a multi-tier application. We compare our work with this body of research in more detail in Section 5. Being completely distributed imparts our algorithm the usual merits associated with such a design, including the lack of high-overhead and complex global synchronization mechanisms, the absence of a single point of failures, and the potential to be highly scalable.

A side-effect of our design is the preferential scheduling of I/O-intensive domains near the beginning of a *period*. This had the beneficial effect of significantly reducing the number of domain context-switches in a consolidated server with the corresponding improvement in efficiency due to fewer address space changes (and associated cache pollution and TLB flushes).

Alternative design choices: A possible pitfall of our approach arises from the fact that it acts based solely on *immediate* reception/transmission intensity. For instance, it preferentially schedules a domain receiving high-intensity traffic even if another domain had received packets earlier. Alternative approaches that incorporate the time that the packets intended for a domain have been waiting in the reception-I/O-ring of a domain also into scheduling decisions are certainly possible. We treat these as outside the scope of our current investigation and intend to evaluate them in our future work. A key issue to appreciate here is that our algorithm continues to provide the CPU reservation guarantees offered by SEDF which ensures that a domain performing high-intensity network activity would not induce an unbounded delay in the scheduling of the other domains, since they are guaranteed their slice time units within their period, only that it could be delayed to the end of their period.

4. Experimental Evaluation

4.1 Experimental setup

Our experimental testbed consists of a pair of Xen hosted servers. Each server has dual Xeon 3.4 GHz CPUs with 2 MB of L1 cache, 800 MHz Front Side Bus, and 2 GB RAM. For our experiments, we pinned all domains to a single CPU. Extending our communication-aware scheduler to multi-processors is part of future work. The machines were setup to use Xen 3.0.2 and are connected via Gigabit Ethernet. Each of our experimental machines hosts 6-8 VMs with each VM assigned between 120 MB to 300 MB of RAM depending on its requirement. Note that although real-world applications are likely to require larger amounts of RAM, they would also be hosted on servers with correspondingly larger memory than is available on the servers in our research lab. Our servers are representative of those used in many hosting platforms except possibly having smaller RAM. We deliberately choose applications whose memory footprints are small enough to allow us to consolidate 6-8 of them on a single machine while meeting their CPU, disk, network, and memory needs. Finally, the applications that we experiment with have CPU and I/O needs that are large enough to make it prohibitive to host hundreds of them on servers employed in data centers as explored in some recent research [43]. Based on these observations, we deem it reasonable to expect degrees of consolidation of at most a few tens of applications. *Domain0* is given 320 MB of RAM. The domains and the physical hosts have unique IPs and the domains communicate via bridge networking.

Applications and workloads: In order to measure improvements in performance and server consolidation we use two applications: (i) a two-tiered implementation of the TPC-W benchmark [34] representing an online bookstore developed at New York University [37], and (ii) a streaming media server. These simulate the kind of real world applications that are likely to be hosted in VHPs. Additionally, we use some domains running CPU-intensive applications for illustrative purposes in some of our experiments.

Our chosen implementation of TPC-W is a fully J2EE compliant application, designed using the “Session Facade” pattern. Session handling is provided by the Apache/ Tomcat servlet container. We configure the application using JBoss 3.2.8SP1 [20] for the application logic tier and MySql 4.1 [25] for the database tier. We use the workload generator provided with TPC-W to simulate multiple concurrent browser clients accessing the application.

We implement a simple, multi-threaded streaming media server and accompanying client in Java. Our streaming media server spawns a thread to stream data to each client. The client program implements a buffer and starts consuming data only when the buffer is filled. If, during a run, the buffer becomes empty (buffer under-run), the client waits till it fills before continuing consumption. Each buffer under-run event is seen as a playback discontinuity and its performance can be measured as a function of how many of these the client experiences while reading streaming data from the server. The server streams data at a constant rate of 3.0 Mbps per client. By default, each client is assumed to use a buffer of size 8 MB. For all experiments, we report statistics collected over 20 minute runs.

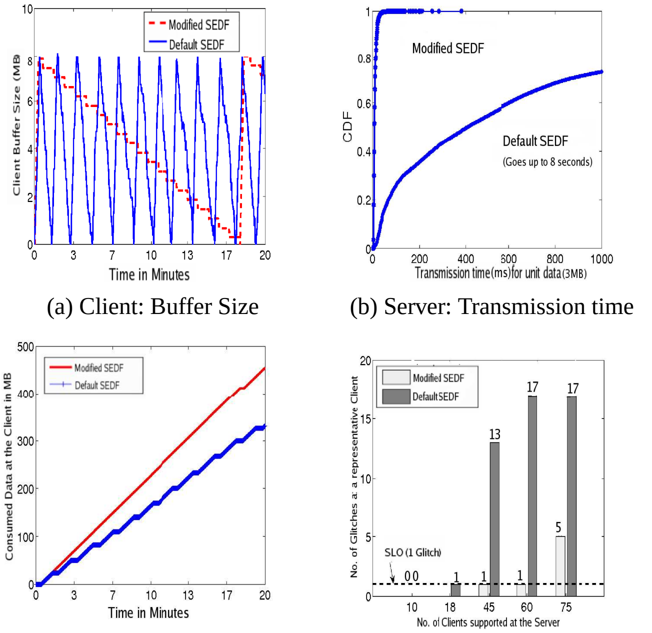
4.2 Performance improvements

To measure the performance benefits due to our scheduler, we run several experiments with TPC-W under various degrees of consolidation and compare the client response times with those on default Xen. Recall that in Figure 1, we showed how consolidating the tiers of a TPC-W application along with several CPU-intensive guest domains on one physical host causes the response times to deteriorate significantly. We present the results of repeating this experiment with our scheduler in Table 1. We find that our scheduler is able to co-ordinate the communication events between the tiers of TPC-W, helping reduce the response times of requests. We report the average, 95^{th} percentile, and maximum values of the observed response times. Similar benefits were found when multiple instances of the TPC-W application are co-located in a single physical server.

Scheduler	Avg. (msec)	95^{th} (msec)	Max. (msec)
Modified SEDF	869	5720	12778
Default SEDF	1319	7149	26158
Percentage Improvement	34.11	19.98	51.15

Table 1. Performance of the TPC-W application.

Next, we investigate the benefits of communication-aware scheduling for our streaming media server. We discuss a representative set of experiments. In this discussion, the domain hosting the media server competes with 7 CPU-intensive domains. Data is streamed to 45 clients at a constant rate of 3.0 Mbps each over a period of 20 minutes. 45 is the maximum number of clients that the server running on the modified hypervisor can support with minimal performance degradation experienced by the clients (one buffer under-run in 20 minutes). We determine this by repeating the experiment with a varying number of clients (reducing the number in each repetition) till the server can support them all to the degree mentioned above. We measure the number of buffer under-runs at each client. We plot this in Figure 6(a) for the default and modified schedulers as noted at a representative client. Whereas with default Xen, a client experiences 11 buffer under-runs on average, our scheduler reduces this number to 1 on average.



(a) Client: Buffer Size

(b) Server: Transmission time

(c) Client: Cumulative data

(d) Server: No. Clients supported

Figure 6. Performance and scalability improvement for streaming media server.

Figure 6(c) shows the cumulative data received by the selected client and depicts the number of times the client experienced discontinuities. Clearly, our scheduler provides much better end-user experience when compared to the default scheduler. Figure 6(b) plots the CDFs for the transmission times of data units (3MB) under the two scheduling policies, allowing us to appreciate the reduction in scheduling-induced delays caused by our scheduler. This graph shows that the enhanced scheduler enables the server to send 95% of its data units under 25 msec, while the default scheduler is able to send 95% of its data units under about 2300 msec, a significant difference in data delivery performance.

Finally, we vary the number of clients and measure their performance to determine any impact our scheduler might have on the scalability of the streaming server. The client buffer size was kept at 6 MB for this set of experiments. Figure 6(d) compares the number of clients for which the streaming server could support an SLO of at most one playback discontinuity during the delivery of a movie. As shown,

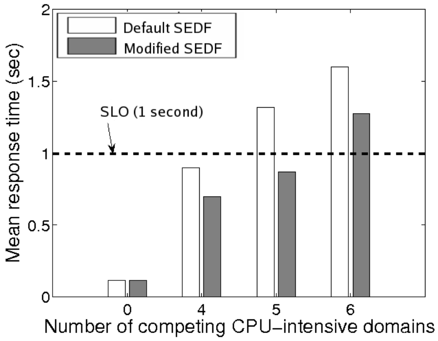


Figure 7. Improved consolidation for TPC-W.

the communication-aware scheduling improves the effective capacity of the streaming server from 18 clients (when hosted on default Xen) to 60 clients (when hosted on our modified hypervisor).

Feature enabled	Time to experience a discontinuity (min)
Only *Domain0* Optimization	14.5
Only anticipation	4
Combination of above two	17

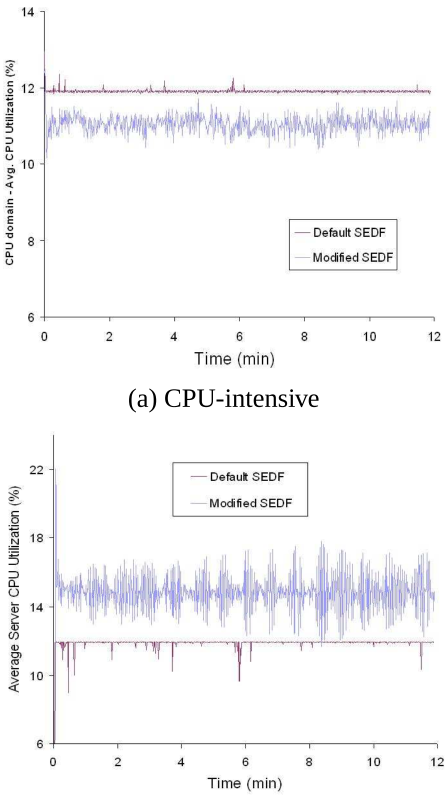
Table 2. Examination of performance with different combinations of our optimizations enabled.

Next, we conduct experiments to ascertain the relative contributions of the various components of our overall scheduling algorithm. Table 2 presents our observations for the streaming media server serving 45 clients. We repeat the experiment thrice, each time with a different combination of our optimizations enabled: (i) only preferential scheduling of *Domain0* , (ii) only anticipatory scheduling, and (iii) both of the above. We use the average time for a client buffer to under-run as the metric and find that preferential scheduling of *Domain0* is crucial to reduce scheduling-induced delays. Anticipatory scheduling, while useful, is ineffective unless *Domain0* is scheduled to complement it. We would like to re-emphasize that merely providing a high CPU reservation of (15, 20) to *Domain0* does not ensure that it gets scheduled at the right times. Our optimization, described in Section 3.4, indeed plays a crucial role in reducing the delays associated with the scheduling of *Domain0*.

4.3 Improved consolidation

We use the same workload for TPC-W as in the last section. We vary the number of CPU-intensive domains consolidated with the tiers of TPC-W from 0 onwards. We pick an SLO of the average response time being 1 sec. Figure 7 shows the results of our experiments.

We find that while with default Xen, we were able to consolidate 4 CPU-intensive domains, with our modified hypervisor, we were able to add an extra CPU-intensive domain, an improvement of 25% in the resulting consolidation. We have observed similar improvements in consolidation for: (i) TPC-W subjected to different workloads and (ii) the streaming media server.



(a) CPU-intensive

(b) Streaming server

Figure 8. Fairness in CPU allocation for CPU-intensive domains consolidated with the streaming media server.

4.4 Evaluation of fairness guarantees

Next, we present another facet of the experiment conducted in Section 4.2 with the streaming media server handling 45 clients. *Do the performance improvements for the streaming media server upon using our scheduling come at the cost of reduced CPU allocations for the competing CPU-intensive domains?* The results presented in Figure 8 address this fairness issue. As seen in Figure 8(a), the CPU-intensive domains continue to receive CPU allocations close to their consumptions on default Xen. There is a decrease of less than 1% in their allocations and our algorithm ensures that they continue to receive CPU more than their reservations. The accumulated 4-5% CPU stolen from these domains is utilized by the streaming media server and *Domain0* to achieve the substantial improvement in performance and scalability as described earlier (see Figure 8(b)).

We conduct similar experiments for evaluating the fairness of CPU allocations with the TPC-W application. We present these results in Table 3. The TPC-W is consolidated with 5 CPU-intensive domains and subjected to the standard TPC-W workload as in Section 4.2. We see that our scheduler is able to match the fairness guarantees provided by default SEDF while improving the performance seen by the clients of TPC-W as described earlier. This is an effective demonstration of the benefit of communication-aware scheduling - our scheduler is providing essentially the same CPU allocation to TPC-W, but by changing the order in which CPU is assigned, significant performance and consolidation gains are being realized. The last column in the table presents the aggregate CPU utilization on the server (nearly identical for both cases).

CPU Usage	*Dom0*	Jboss	DB	CPU.	Agg.
Def.(mean)	0.65	1.96	4.43	17.58	94.94
Def.(variance)	0.13	11.57	40.07	3.30	
Mod.(mean)	0.65	1.68	4.53	17.89	96.31
Mod.(variance)	0.17	10.35	48.31	3.38	

Table 3. Average CPU utilization (%) for different domains running TPC-W with the default and the modified schedulers. *Legend:* Def.=Default SEDF; Mod.=Modified SEDF; CPU.=One among the 5 CPU-intensive VMs; Agg.=Aggregate CPU usage of the Server.

4.5 Reduced context switching

In Section 3.5, we had hypothesized that our scheduler may have the beneficial side-effect of reducing the overall domain context switches by coalescing the scheduling of communication-intensive domains towards the beginning of a *period*. We conduct measurements to validate this using XenMon [18]. For a server hosting our streaming server with 7 CPU-intensive domains, we found that the number of domain context switches were reduced by almost 33% when using our modified hypervisor compared to default Xen. We postulate that with more consolidated communicating tiers, we might see a further reduction in the number of context switches.

5. Related Work

Earlier , we pointed out aspects of existing research on non-virtualized hosting platforms relevant to the design of a VHP. In this section, we discuss additional existing research on virtualization and scheduling that is closely related to this paper.

Efficient virtualization techniques As mentioned earlier, virtualization is being actively researched and employed for designing consolidated hosting environments [11, 44, 8]. Consequently, there have been several prior studies proposing and evaluating novel enhancements to virtualization platforms such as Xen and VMware [41, 23]. Another relevant body of work is concerned with reducing the overheads of virtualization [24] and accounting them accurately to hosted VMs to ensure fair CPU allocations [17, 18, 16].

Scheduling in parallel systems The need for co-ordinated scheduling of communicating entities has been extensively looked at in the context of parallel applications running on tightly coupled multi-processors (whether it be message-passing or shared-memory systems as well as clusters. While earlier work attempted this by explicitly performing periodic synchronization [22, 45], subsequent relaxations explored the possibility of local scheduling at each node based on communication events to achieve similar goals [3, 35, 26, 36, 46]. While the goals of our work are similar, to the best of our knowledge, this is the first study to explore these ideas in the domain of the Xen VMM for multi-tier applications. These applications have unique characteristics including being more loosely coupled than the ones previously studied and tiers with heterogeneous resource needs. Additionally, VHPs are likely to support substantially higher levels of consolidation/multi-programming at each node than the platforms studied in earlier work. Finally, over and beyond metrics such as throughput and overall completion time that the traditional parallel systems try to optimize, VHPs are expected to provide responsiveness and fairness guarantees as well. Our scheduling mechanisms attempt to address these multiple goals.

6. Conclusions

Advances in virtualization technologies have created a lot of interest among Internet data center providers to exploit features of VMMs for cost-cutting via improved consolidation. We identified one major shortcoming in the Xen VMM that proves to be an obstacle in its efficient operation when employed in a VHP: The VM scheduler in Xen is agnostic of the communication behavior of modern, multi-tier applications and also the scheduling of the privileged domain is in the critical path of every network operation.

We developed a new communication-aware CPU scheduling algorithm for the Xen VMM. Using experiments with realistic Internet server applications and benchmarks, we demonstrated the performance/cost benefits and the wide applicability of our algorithms. For example, the TPC-W benchmark exhibited improvements in average response times of up to 35% for a variety of consolidation scenarios. A streaming media server hosted on our prototype VHP was able to satisfactorily service up to 3.5 times as many clients as one running on the default Xen. The source code for our implementation is publicly available.

Acknowledgements

We would like to thank the anonymous reviewers for their detailed comments which helped us improve the quality of the presentation. This research was funded in part by NSF grants 0103583, 0509234, 0325056 and 0429500. Dr.Bhuvan Urgaonkar would also like to acknowledge a start-up grant provided by the CSE Department, Pennsylvania State University to support his research.

References

[1] M. Adler, Y. Gong, , and A. Rosenberg. Optimal Sharing of Bags of Tasks in Heterogeneous Clusters. In *Proceedings of the Fifteenth Annual ACM*

Symposium on Parallel Algorithms and Architectures (SPAA), Scheduling I, pages 1–10. ACM Press, 2003.

[2] Amazon Elastic Compute Cloud. `http://www.nature.com/`.

[3] A. Arpaci-Dusseau. Implicit Coscheduling: Coordinated Scheduling with Implicit Information in Distributed Systems. *ACM Transactions on Computer Systems*, 19(3):283–331, 2001.

[4] A. Arpaci-Dusseau and D.E. Culler. Extending Proportional-Share Scheduling to a Network of Workstations. In *Proceedings of Parallel and Distributed Processing Techniques and Applications (PDPTA'97), Las Vegas, NV*, June 1997.

[5] G. Banga, P. Druschel, and J. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Proceedings of the Third Symposium on Operating System Design and Implementation (OSDI'99), New Orleans*, pages 45–58, February 1999.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebuer, I. Pratt, and A. Warfield. Xen and the Art of Virtulization. In *Proceedings of the Nineteenth Symposium on Operating Systems Principles (SOSP)*, 2003.

[7] M. Benani and D. Menasce. Resource Allocation for Autonomic Data Centers Using Analytic Performance Models. In *Proceedings of IEEE International Conference on Autonomic Computing, Seattle (ICAC-05), WA* , June 2005.

[8] M. Bennani and D. Menacse. Autonomic Virtualized Environments. In *Proceedings of the IEEE International Conference on Autonomic and Autonomous Systems (ICAS 2006), Santa Clara, CA*, July 2006.

[9] A. Chandra, M. Adler, P. Goyal, and P. Shenoy. Surplus Fair Scheduling: A Proportional-Share CPU Scheduling Algorithm for Symmetric Multiprocessors. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI 2000), San Diego, CA*, October 2000.

[10] Y. Chen, A. Das, Q. Wang, A. Sivasubramaniam, R. Harper, and M. Bland. Consolidating Clients on Back-end Servers with Co-location and Frequency Control. In *Posthe ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2006), June 2006*, June 2006.

[11] C. Clark, K. Fraser, Steven Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005.

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

[13] R. Doyle, J. Chase, O. Asad, W. Jin, and Amin Vahdat. Model-Based Resource Provisioning in a Web Service Utility. In *Proceedings of the Fourth USITS*, March 2003.

[14] 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 Seventeenth ACM Symposium on Operating Systems Principles*, pages 261–276, New York, NY, USA, 1999. ACM Press.

[15] R. Goldberg. Survey of Virtual Machine Research. *IEEE Computer*, pages 34–45, June 1974.

[16] S. Govindan, A. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. Communication-aware CPU Management in Consolidated Virtualization-based Hosting Platforms. Technical report, Department of Computer Science and Engineering, The Pennsylvania State University, October 2006.

[17] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing Performance Isolation Across Virtual Machines in Xen. In *Proceedings of the Seventh International Middleware Conference, Melbourne, Australia*, November-December 2006.

[18] D. Gupta, R. Gardner, and L. Cherkasova. XenMon: QoS Monitoring and Performance Profiling Tool. Technical Report HPL-2005-187, HP Labs, 2005.

[19] Intel VT. `http://www.intel.com/technology/itj/2006/v10i3/foreword.htm`.

[20] The JBoss Application Server. `http://www.jboss.org`.

[21] M. B. Jones, D. Rosu, and M. Rosu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles (SOSP'97), Saint-Malo, France*, pages 198–211, December 1997.

[22] S. T. Leutenegger and M. K. Vernon. The Performance of Multiprogrammed Multiprocessor Scheduling Algorithms. In *SIGMETRICS '90: Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 226–236, 1990.

[23] J. Liu, W. Huang, B. Abali, and D. K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. In *Proceedings of the USENIX Annual Technical Conference (USENIX'06), Boston, MA*, May-June 2006.

[24] A. Menon, A. Cox, and W. Zwaenepoel. Optimizing Network Virtualization in Xen. In *Proceedings of the USENIX Annual Technical Conference (USENIX'06), Boston, MA*, May 2006.

[25] MySQL. `http://www.mysql.com`.

[26] S. Nagar, A. Banerjee, A. Sivasubramaniam, and C. R. Das. A Closer Look at Co-scheduling Approaches for a Network of Workstations. In *SPAA '99: Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures*, pages 96–105, 1999.

[27] M. Nelson, B. Lim, and G. Hutchins. Fast Transparent Migration for Virtual Machines. In *Proceedings of the 2005 USENIX Annual Technical Conference*, pages 391–394, April, 2005.

[28] J. Nieh and M. Lam. A SMART Scheduler for Multimedia Applications. *ACM Transactions on Computer Systems*, 21(2):117–163, 2003.

[29] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of Fifth USENIX Symposium on Operating Systems Design and Implementation*, pages 361–376, 2002.

[30] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the Migration of Virtual Computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[31] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated Resource Management for Cluster-based Internet Services. In *Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation (OSDI), Boston, MA*, December 2002.

[32] S. Shende, A. Malony, J. Cuny, K. Lindlan, P. Beckman, and S. Karmesin. Portable Profiling and Tracing for Parallel Scientific Applications using C++. In *Proceedings of ACM SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT)*, pages 134–145, August 1998.

[33] J. E. Smith and R. Nair. *Virtual Machines: Architectures, Implementations and Applications*. Morgan Kauffmann, New York, 2004.

[34] W. Smith. TPC-W: Benchmarking An Ecommerce Solution. `http://www.tpc.org/information/other/techarticles.asp`.

[35] P. Sobalvarro and W. E. Weihl. Demand-Based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors. In *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 106–126, 1995.

[36] M. S. Squillante, Y. Zhang, A. Sivasubramaniam, N. Gautam, H. Franke, and J. Moreira. Modeling and Analysis of Dynamic Co-scheduling in Parallel and Distributed Environments. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 43–54, 2002.

[37] NYU TPC-W. `http://www.cs.nyu.edu/pdsg/`.

[38] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An Analytical Model for Multi-tier Internet Services and its Applications. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2005), Banff, Canada*, June 2005.

[39] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. In *Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA*, December 2002.

[40] VMware. `http://www.vmware.com/`.

[41] C. Waldspurger. Memory Resource Management in VMWare ESX Server. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI'02)*, December 2002.

[42] C. A. Waldspurger and W. E. Weihl. Lottery Scheduling: Flexible Proportional-share Resource Management. In *Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI'94)*, November 1994.

[43] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI'02)*, December 2002.

[44] XenSource Press Release. `http://www.xensource.com/news/pressreleases.html`.

[45] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam. Improving Parallel Job Scheduling by Combining Gang Scheduling and Backfilling Techniques. In *IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing*, pages 133–142, 2000.

[46] Y. Zhang, A. Sivasubramaniam, J. E. Moreira, and H. Franke. A Simulation-based Study of Scheduling Mechanisms for a Dynamic Cluster Environment. In *Proceedings of the 11th ACM International Conference on Supercomputing (ICS)*, pages 100–109, 2000.