

Communication-aware CPU Management for Consolidated Virtualization-based Hosting Platforms

Sriram Govindan
sgovinda@cse.psu.edu

Arjun R. Nath
anath@cse.psu.edu

Amitayu Das
adas@cse.psu.edu

Bhuvan Urgaonkar
bhuvan@cse.psu.edu

Anand Sivasubramaniam
anand@cse.psu.edu

*Department of Computer Science and Engineering,
The Pennsylvania State University, University Park, PA, 16802.*

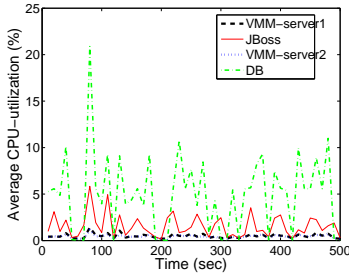
October 17, 2006

Abstract

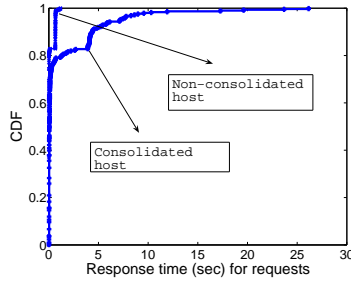
Recent advances in software and architectural support for server virtualization have created interest in using this technology in the design of consolidated hosting platforms. Since virtualization enables easier and faster application migration as well as secure co-location of antagonistic applications, higher degrees of server consolidation are likely to result in such virtualization-based hosting platforms (VHPs). We identify two shortcomings in existing virtual machine monitors (VMMs) that prove to be obstacles in operating hosting platforms, such as Internet data centers, under conditions of such high consolidation: (i) CPU schedulers that are agnostic to the communication behavior of modern, multi-tier applications and (ii) inadequate or inaccurate mechanisms for accounting the CPU overheads of I/O virtualization. We develop a new communication-aware CPU scheduling algorithm and a CPU usage accounting mechanism. We implement our algorithms in the Xen VMM and build a prototype VHP on a cluster of 36 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 between 20%-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 Introduction and Motivation

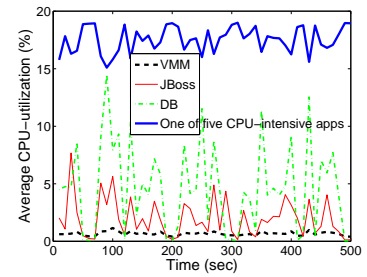
The recently resurgent research in server virtualization has fuelled interest in using this technology to design consolidated hosting platforms. In this emerging hosting model, each physical server in the cluster runs a software layer called the Virtual Machine Monitor (VMM) that virtualizes the resources 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



(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.

overlying operating systems. The development of highly efficient VMMs [56, 6, 53] as well as the evolution of architectural support for them [24] 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 untrusting services [42, 32, 17, 41].

Cluster-based hosting platforms have received extensive attention in several research communities such as those dealing with operating systems [5, 13, 52, 34, 43], parallel/distributed computing [4, 46, 36, 47, 60], 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 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 systems mechanisms to address two such issues.

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. Server virtualization enables untrusting applications (as well as applications written for heterogeneous operating systems) to be securely co-located. Furthermore, it facilitates faster migration of application components across physical servers and enables rapid addition or removal of servers to/from the pool assigned to a hosted application [51, 32, 42]. Ensuring that the applications experience satisfactory performance even under such consolidation

¹Note 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.

requires the hosting platform to perform (i) *resource requirement estimation* for the hosted applications, done either by application profiling [44, 2, 52] or using analytical models [40, 18, 8, 49] 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) [13, 14, 52, 43, 50]. 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 a TPC-W benchmark 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. *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*.

Problem 1: *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?*

The need for accurate accounting of the overheads of virtualization: The introduction of the VMM layer substantially changes the problems of resource usage book-keeping and accounting, particularly for the CPU. In a VHP, the fundamental unit of resource allocation and accounting changes from an application or a process to a VM. Additionally, the VMM is responsible for virtualizing I/O devices, which manifests itself in the form of CPU overheads. In hosting platforms where applications are paying for the resources, it is important to accurately assign these overheads of virtualization to the VMs they originate from. This payment may be explicit, as in Internet data centers, or implicit, as in an enterprise data center hosting applications serving multiple “equally important” departments internal to an organization. Internet data centers, for example, host disk-intensive database servers or network-intensive streaming servers. Hosting such applications can

result in high CPU overheads for I/O virtualization. In the absence of accurate mechanisms for accounting these CPU overheads to the applications they originate from, unfair CPU allocations are likely to result, an undesirable situation in the hosting platforms that we are interested in.

Problem 2: Can a server in a VHP account for the overheads of virtualization and incorporate them into its CPU scheduling to provide fair resource allocations?

1.1 Research contributions

Our research contribution is threefold. First, 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.

Second, we develop an algorithm that accurately and efficiently accounts the CPU overheads resulting from server virtualization and attributes these to the VMs they originate from. We then incorporate these measurements into our CPU scheduling algorithm, enabling it to provide more meaningful CPU guarantees to hosted applications. Our accounting technique also relieves the administrators of a VHP of having to anticipate and explicitly provision CPU capacity for the I/O virtualization to be performed by the VMM. This is an improvement because it eliminates the non-determinism in CPU allocations that results from the ad-hoc ways in which this “knob” is set in existing systems.

Finally, we identify efficient ways of implementing these algorithms in the state-of-the-art Xen VMM. We use the Xen VMM, enhanced with our algorithms, to build a prototype VHP of 36 physical servers. We explore the pros and cons of our implementation by experimenting with realistic and representative Internet server 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 in the range 20%-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 virtualization-based hosting platforms in Section 2. We discuss the design and implementation of our communication-aware CPU scheduling and accurate accounting of the CPU usage of virtually hosted applications in Sections 3 and 4, respectively. We describe our experimental setup and evaluation in Section 5. We present related work in Section 6. Finally, we present concluding remarks in Section 7.

2 Background and System Overview

In this section, we provide an overview of our virtualized hosting model.

2.1 The Xen VMM

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 or 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 a server. A VMM virtualizes the resources of a physical server and supports the execution of multiple virtual machines (VMs) [20, 6, 45, 56]. 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 [20, 21, 42, 7, 45, 6, 17, 32, 38]. These benefits come at a price, however. Any virtualization technique has associated CPU overheads such as those due to I/O virtualization.

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 *Domains* in Xen) each. One of the VMs is “privileged”, similar to that present in the Xen VMM. The privileged VM (called *Domain0*) implements the real device drivers and does the translation between virtual and real I/O activity. The CPU activity involved in this translation forms part of the overhead presented by virtualization.

I/O virtualization in Xen: Our description is specific to network virtualization in Xen. Disk I/O virtualization is realized in a similar manner. 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 arrived 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.

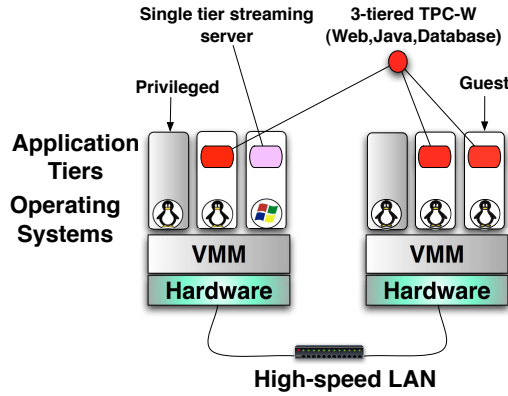


Figure 2: Illustration of hosting in a VHP.

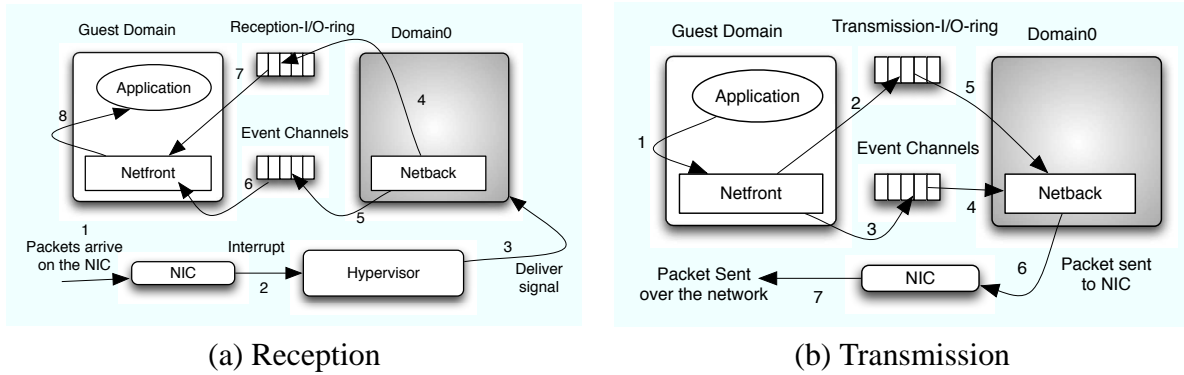


Figure 3: Network I/O virtualization in Xen.

2.2 A virtualized hosting platform

Our hosting model assumes a large cluster of high-end servers (say 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. Our enhanced hypervisor implements: (i) a modified CPU scheduler to achieve communication-aware scheduling of domains, and (ii) mechanisms that measure and maintain relevant resource usage statistics pertaining to the overlying domains, that are used by our scheduler. Figure 2 provides an illustrative example.

3 Communication-aware Scheduling in a VHP

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 [19, 55, 12, 26, 30, 37]. 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 [52, 43]. 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 desire². 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.

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 2.1 and Figure 3, we identify three sources of scheduling-induced delays.

1. **Delay at the recipient:** This is the duration between when a network packet arrives for a domain and when it copies this packet from the reception-I/O-ring in *Domain0* into its own address space upon getting scheduled next. This delay can be reduced by scheduling a domain soon after the reception of a packet for it.
2. **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.

²We 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.

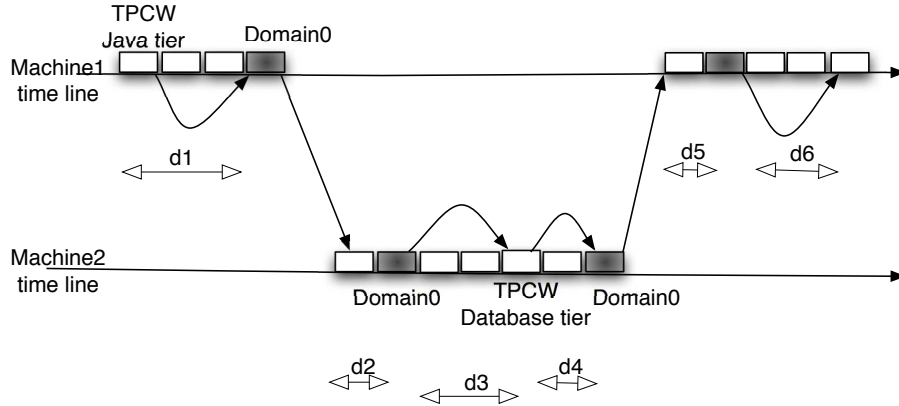


Figure 4: Sources of scheduling-induced delays.

3. **Delay associated with the scheduling of *Domain0*:** This is either (i) the duration between a packet reception at the physical network interface and when *Domain0* copies it into the address space of the domain the packet was addressed to or (ii) the duration between when a domain copies a packet into the transmit-I/O ring of *Domain0* and when *Domain0* gets scheduled next to actually send it over the network interface. These delays can be reduced by (i) scheduling *Domain0* soon after a packet reception for a domain occurs and (ii) soon after a domain does a send operation over its virtual network interface, respectively.

With these intuitions, we now consider ways to reduce each of these three kinds of delays 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 $d3$ and $d6$ are of type 1, while delays $d1$, $d2$, $d4$, and $d5$ are of type 3.

3.2 Preferential scheduling of recipient

Scheduling a domain close to the reception of a packet for it is easy to achieve in theory, since this is a purely reactive act. 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?* A naturally appealing heuristic is one 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*. We leave this *period* as a tunable parameter for the platform administrator to choose.

Implementation considerations: In Xen, each domain 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 packets³ received by that domain and increments the *network-reception-intensity* variable of the corresponding recipient domain in the book-keeping 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 we use the number of packets sent by a domain during its last transmission 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 it 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-

³This 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 [33].

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.

3.4 Scheduling of *Domain0*

As depicted in Figures 3(a) and (b), *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*.

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.

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 *anticipated-network-transmission-intensity* for *Domain0*. Figure 5 presents an example to help understand the overall implementation of the scheduler.

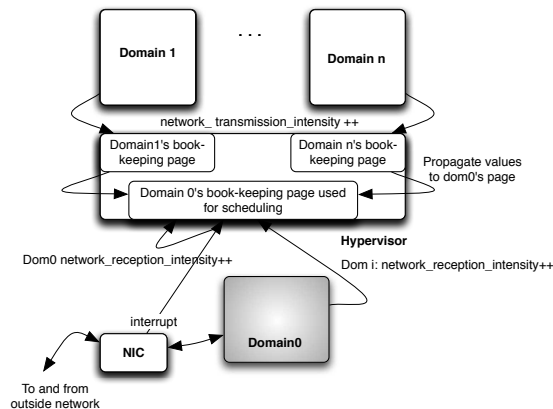


Figure 5: Implementation of our scheduler.

3.5 Salient features and alternatives

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

Fairness issues: There are two facets of fair CPU allocation in a consolidated server in our VHP. First, we require our scheduler to guarantee that each domain receives at least its specified *slice* during every *period*. It is easy to see that our algorithm meets this requirement. The condition *Respect Reservations* described earlier ensures that our algorithm changes only the scheduling order of domains within a *period*; the CPU allocations provided to each domain within a *period* remain the same as in the default SEDF algorithm.

The second aspect of fairness is concerned with accounting of the CPU costs that are incurred in a VHP for virtualizing the I/O activities of hosted domains. We motivate and address this issue in Section 4.

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 6. 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 is likely to have the beneficial effect of 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). We experimentally demonstrate a scenario where such benefits are evident in Section 5.

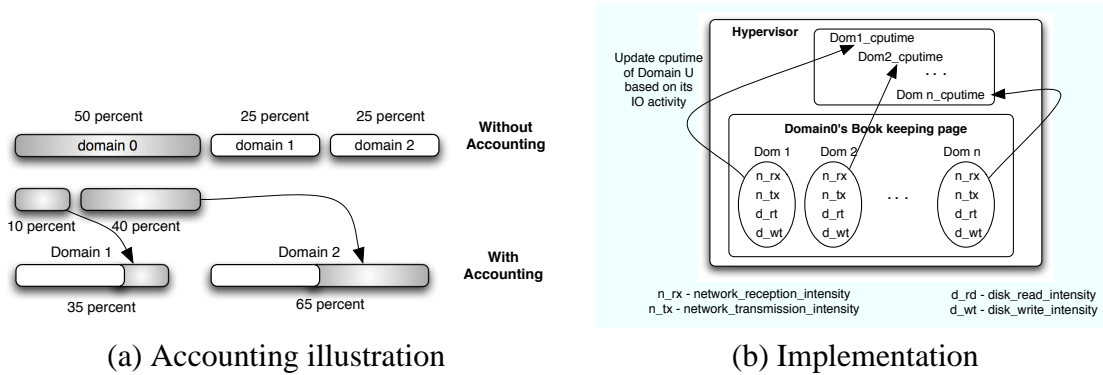


Figure 6: Accounting of I/O virtualization overheads.

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. Such behavior can result in deviations from the goal of the algorithm which is to minimize the sum of scheduling-induced delays. 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 the CPU reservation guarantees offered by SEDF that our algorithm continues to provide ensure that a domain receiving high-intensity traffic would not induce an unbounded delay in the scheduling of other domains since it can be preferentially scheduled only as long as it has used less than its *slice* within the ongoing *period*.

4 Accounting of I/O-Virtualization Overheads

When I/O-intensive tiers are consolidated onto a server, *Domain0* is likely to become highly CPU-intensive. This is analogous to increasing time spent in the kernel mode by a computer when the overlying applications perform high-intensity I/O. In a VMM such as Xen, most of the functionality related to I/O virtualization resides in *Domain0*. Consequently, a majority of the associated CPU overhead can be attributed to *Domain0*. We assume that the CPU activity performed by the hypervisor is small enough that it does not interfere with the CPU activities or needs of the guest domains in any discernible way. This has been observed by other researchers [6, 16] and our experimental evaluation in Section 5 also confirms this. Existing research on operating systems has identified the necessity to correctly account the aforementioned CPU usage on the part of the kernel to its original sources, namely the applications whose I/O activities are responsible for it [5]. *Are similar accounting mechanisms needed in the VMMs in our VHP?* We believe the answer is yes.

To motivate this, let us consider two general kinds of CPU scheduling algorithms that a VHP may use to provide CPU guarantees to guest domains: proportional-share (PS) [55, 19, 39] and reservation-based (such as the SEDF algorithm employed by Xen) schedulers [37, 26]. Generally speaking, as guest domains perform more I/O, the fraction of CPU available to them de-

creases, since an increasing fraction of it is consumed by *Domain0*. Consequently, employing a PS scheduler would clearly result in a shrinkage of the CPU capacity available to an individual guest domain. An I/O-intensive domain could adversely affect the performance of other domains, an example of the *QoS cross-talk* that some operating systems research has addressed [28].

For reservation-based schedulers like SEDF, that provide *absolute* lower bounds on the CPU allocations for domains, the problem is of a different nature. Since *Domain0* is a schedulable entity similar to other guest domains, these schedulers require a system administrator to assign a lower bound on its CPU reservation. This automatically reduces the degree of consolidation that can be achieved. For example, it is recommended that with the SEDF scheduler, *Domain0* be given a reservation of (15 msec, 20 msec), implying that the aggregate CPU reservation for the guest domains can not exceed (5 msec, 20 msec)⁴. As a consequence, a domain with high I/O intensity can receive a larger *effective* share of the CPU in the form of the activity performed on its behalf by *Domain0*. In case there is residual CPU capacity (due to one or more domains not using its entire reservation), it is distributed to the backlogged domains in a round-robin fashion, an unsatisfactory solution in a VHP where hosted applications desire more concrete resource guarantees. In the rest of this section, we identify the cause of this problem and develop a solution for it.

4.1 Our accounting mechanism

The central problem is that domains that avail of the services of *Domain0* for their I/O activities are not *charged* for these services. Ideally, we would like for each guest domain to specify its CPU reservation *including the virtualization costs of its I/O needs*. Next, we would like to be able to keep an accurate account of the time spent by *Domain0* on behalf of each of the guest domains. As mentioned in Section 3, the first of these requirements is orthogonal to our research. In this paper, we focus on realizing the second requirement of accounting. Having these two mechanisms would solve the various problems outlined earlier. First, it would relieve the administrator from having to separately provision CPU for *Domain0*, which as argued already, is done in an ad-hoc fashion. Second, (as a result of the previous point) it would allow better consolidation by not statically tying up a fixed fraction of the CPU capacity for *Domain0*. Finally, it would help fix the non-determinism in the scheduler behavior alluded to earlier and achieve more fair and predictable CPU allocations.

Our approach involves a simple enhancement to our CPU scheduler. After the n^{th} execution of *Domain0* (starting the count at, say, system start up), we partition the time it ran during this execution, d_0^n , into times $d_{0,1}^n, \dots, d_{0,m}^n$ spent on behalf of each of the m existing guest domains; $d_0^n = \sum_{i=1}^m d_{0,i}^n$. These times are then added to the execution times of the respective guest domains maintained by the scheduler. The rest of the scheduler remains unmodified. Figure 6(a) illustrates this. Notice that *Domain0* does not have an explicitly specified CPU reservation now. Our algorithm, developed in Section 3, ensures that *Domain0* continues to get preferentially scheduled whenever it is on the critical path for crucial activities such as interrupt processing. By charging each guest domain in the fashion described above, our scheduler is able to provide guarantees on the CPU usage of a *guest domain and what Domain0 does for it* rather than only for the guest domain and avoid leaving the sharing of CPU at the mercy of the (possibly dynamically chang-

⁴For the actual implementation of SEDF in Xen, we found this not be strictly true - the aggregate reservation of guest domains is allowed to exceed (5, 20) while the reservation for *Domain0* is (15, 20). We implement a check in SEDF to ensure lower bounds are indeed provided.

ing) relative I/O-intensities of the hosted guest domains. We next describe specific implementation details to achieve this accounting in Xen.

4.2 Implementation considerations

A major proportion of the CPU usage of *Domain0* originates from the virtualization of network and disk I/O. We classify I/O into four kinds: network receptions/transmissions and disk reads/writes. We conduct four separate profiling experiments, one for estimating the CPU overhead of virtualizing of each of these four kinds of I/O. In each of these experiments, we record the time spent by *Domain0* to perform several instances of the corresponding I/O activity. From this, we estimate the average time spent by *Domain0* for virtualizing each activity: δ_{net-rx} , δ_{net-tx} , δ_{disk-r} , and δ_{disk-w} . The quantities $d_{0,i}^n$ are now easily determined by recording the number of page-flips for each type of I/O for every domain and multiplying these by the appropriate δ_* values. We introduce variables in the data structures used by the hypervisor to maintain these counts and change the scheduler code to update the execution times of the domains as described earlier. Figure 6(b) presents the gist of our implementation.

5 Experimental Evaluation

5.1 Experimental setup

Our experimental testbed consists of a rack of 36 servers. Each server has dual Xeon 3.4Ghz CPUs with 2MB of L1 cache, 800MHz Front Side Bus, and 2GB RAM. The machines are connected via a gigabit Ethernet. Our experimental machines host between 8 and 12 domains with each domain being assigned between 120 to 300MB of RAM depending on its requirement. *Domain 0* is given 320MB of RAM. The domains and the physical hosts have unique IPs and domains communicate via bridge networking. All machines are setup to boot either Fedora Core FC4 (SMP) or Xen 3.0.2.

Server applications: In order to measure improvements in performance and server consolidation we use two applications: (i) TPC-W-NYU [48], a three-tiered application based on the TPC-W benchmark for an online bookstore, 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.

TPC-W-NYU is a fully J2EE compliant application, designed using the “Session Facade” pattern. Session handling is provided by the Apache Tomcat servlet container. We classify the requests served by this application into three classes based on their service times and denote these as $r1$, $r2$, and $r3$, respectively. We setup the application using JBoss 3.2.8SP1 [25] as the middle tier platform and MySQL 4.1 [35] for the database tier. We used the workload generator provided with TPC-W-NYU 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 performance can be measured

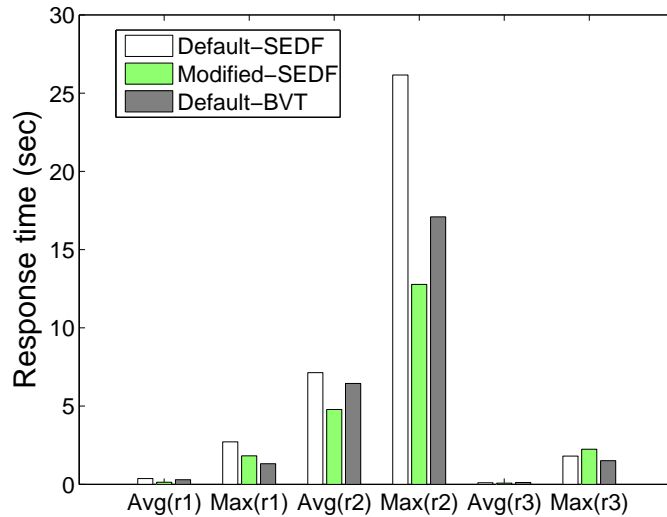


Figure 7: Performance improvement for TPC-W.

as a function of how many of these the client experiences while reading streaming data from the server. By default, each client was assumed to use a buffer of size 8 MB.

5.2 Performance improvements

To measure the performance benefits due to our scheduler, we run several experiments with TPC-W-NYU 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 Figure 7. We find that our scheduler is able to co-ordinate the communication events between the tiers of the TPCW application, helping reduce the average and maximum response times of requests. In particular, for the requests of type r_2 , there is a 25% improvement in average response time and almost 50% improvement in the worst-case response time. We also present the response times offered by a heuristic called low-latency dispatch implemented in Xen to work with BVT, a PS scheduler, for preferentially scheduling communicating domains. The response time improvements due to our scheduler are found to be better than this.

We next turn to investigating 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. We measure the number of buffer under-runs and the cumulative data received at each client. We plot these in Figures 8(a) and (c), respectively, for default Xen and with our modified scheduler 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. Figure 8(c) shows the cumulative data received by the selected client and depicts the number of times the client experienced discontinuities. Clearly, our scheduler

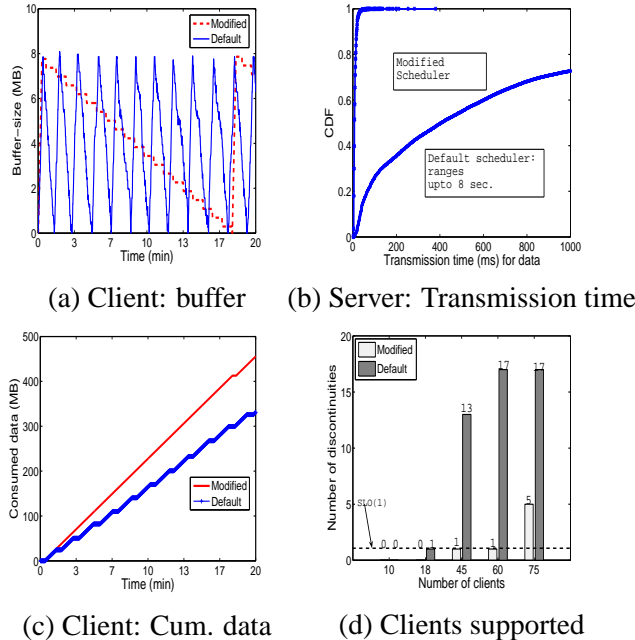


Figure 8: Performance and scalability improvement for streaming media server.

provides much better end-user experience when compared to the default scheduler. Figure 8(b) plots the CDFs for the transmission times of data units 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 8(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, the communication-aware scheduling improved the effective capacity of the streaming server from 18 clients (when hosted on Xen) to 60 clients (when hosted on the modified hypervisor).

Next, we conducted experiments to ascertain the relative contributions of the various components of our overall scheduling algorithm. Table 1 presents our observations for the streaming media server serving 45 clients. We repeated the experiment thrice, each time with a different combination of our optimizations enabled: (i) only *Domain0* related, (ii) only anticipatory scheduling, and (iii) both of the above. Note that for the streaming server, it is easy to see that the third optimization (recipient related) is not worth pursuing, since all the communication is one-way (that is, server to clients). 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.

Feature enabled	Time to experience a discontinuity (min)
Only <i>Domain0</i> 's reception	14.5
Only anticipation	4
Combination of above two	17

Table 1: Examination of performance with different combinations of our optimizations enabled.

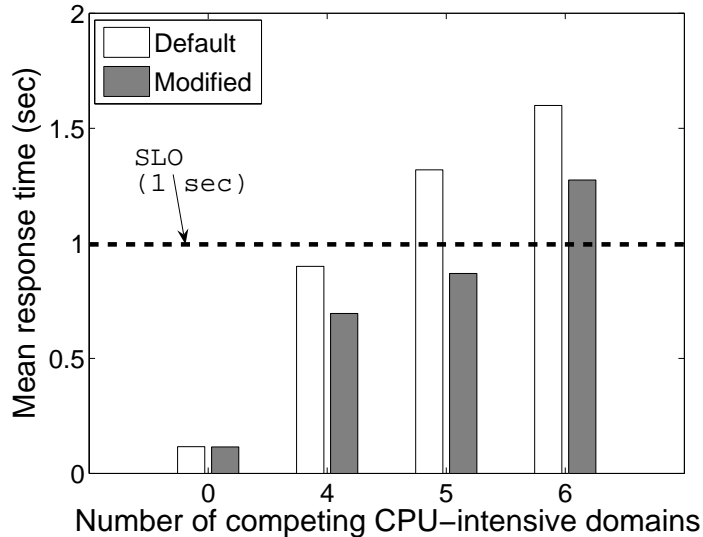


Figure 9: Improved consolidation for TPC-W.

5.3 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 9 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 different workloads as well as for the streaming media server.

5.4 Evaluation of fairness guarantees

Next, we present another facet of the experiment conducted in Section 5.2 with the streaming media server handling 45 clients. *Did the performance improvement 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 10 address this fairness issue.

As seen in Figure 10(a), the CPU-intensive domains continue to receive CPU allocations close to their consumptions on default Xen. There is a decrease of 1% in their allocations and our algorithm ensures that they continue to receive CPU more than their reservations. The accumulated 5-7% CPU stolen from these domains is utilized by the streaming media server and *Domain0* to achieve the substantial improvement in performance and scalability that we described earlier (see

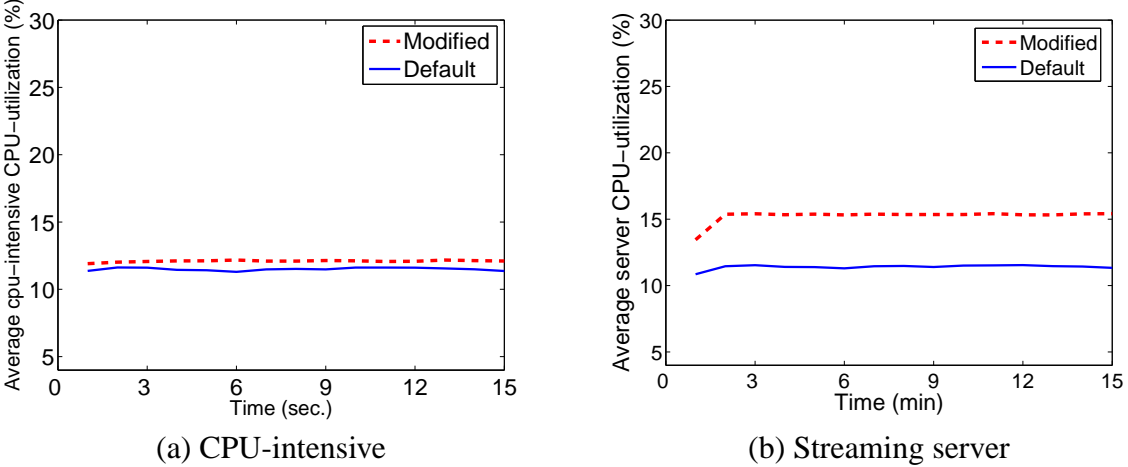


Figure 10: Fairness in CPU allocation for CPU-intensive domains consolidated with the streaming media server.

Figure 10(b)).

Similar experiments were conducted for evaluating the fairness of CPU allocations with the TPC-W application. We present these results in Table 2. The TPC-W was consolidated with 5 CPU-intensive domains and subjected to the standard TPC-W workload as in Section 5.2. 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.

Metric	Domain0	Jboss	DB	CPU-intensive
Default (mean)	0.65	1.96	4.43	17.58
Default (variance)	0.13	11.57	40.07	3.30
Default (max)	2.41	16.01	15.87	19.08
Modified (mean)	0.65	1.68	4.53	17.89
Modified (variance)	0.17	10.35	48.31	3.38
Modified (max)	2.74	18.18	25.20	19.58

Table 2: Average CPU utilization (%) for different domains with the default and the modified schedulers.

5.5 Evaluation of accounting

There are two main problems with the existing mechanism in Xen which can cause unfairness and inconveniences. First, non-reconciling the time that *Domain0* spends on behalf of each hosted domain can cause a domain without significant I/O activity to unfairly suffer. This is because the domain with higher I/O intensity can take a higher CPU fraction (by exercising *Domain0*). Second, the existing mechanism requires the administrator to explicitly specify the required reservation for *Domain0*, which needs to be at least as much as the sum of the I/O virtualization needs of the hosted domains. Our proposed accounting mechanism provides solutions to both these problems. We account for the time that *Domain0* spends on behalf of each guest domain and accordingly

subtract the required CPU reservation specified by the administrator for these domains. Due to our charging the time spent by *Domain0* to domains in proportion to their I/O, a less I/O-intensive domain is not penalized. Further, one could choose to specify an arbitrarily low value for the initial reservation of *Domain0* and our accounting mechanism would automatically adapt it to the exact I/O needs of the different domains.

To illustrate these features, we run an experiment under high CPU load conditions (i.e., sum of CPU reservations is close to 100%) with both TPC-W tiers, the streaming media server, and a CPU-intensive domain. The reservation for *Domain0* is also kept quite low (1.5, 20), relative to the I/O needs. Table 3 compares the two executions, with and without accounting. Between the two I/O demanding loads, the media server is much more communication-intensive than TPC-W. As we can see from the table, the response time for TPC-W is increased three-fold because of this interference from the media server compared to reducing the utilization of the media server based on the fraction of time *Domain0* spends on its behalf. As we can see from the 95th percentile of CPU utilization, the CPU received by the media server reduced by 6% upon using accounting. This newly created CPU slack is being distributed to *Domain0* (whose 95th percentile increased by over 15%) and other domains. This helps to considerably improve the response time of TPC-W clients while not affecting the media server performance. Note that the cumulative data received by a representative client of the media server shows no tangible degradation with our accounting.

Accounting?	TPC-W	Media		VMM
	(1,20)	(4,20)		(1.5,20)
	Resp-time (ms)	95% CPU	Cum. data	95% CPU
No	647	27.70	227MB	14
Yes	213	21.0	226MB	30.68

Table 3: Demonstration of the working of our accounting mechanism.

5.6 Reduced context switching

In Section 3.5, we have 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 [23]. 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 communicating tiers consolidated on this server, we might see a further reduction in the number of context switches.

6 Related Work

Earlier in this paper, we pointed out several aspects of existing research on non-virtualized hosting platforms that are relevant to the design of a VHP. In this section, we discuss additional existing research on virtualization, scheduling, and accounting 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 [17, 58, 9]. Consequently, there have been several prior studies proposing and evaluating novel enhancements to virtualization platforms such as Xen and VMware [54, 33, 31]. However, most of this prior work is largely

restricted to a single VMM hosting multiple applications. Our focus in this work is more on optimizing a pool of these VMMs running on different machines that host multi-tier applications.

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 [11]) as well as clusters. While earlier work attempted this by explicitly performing periodic synchronization [29, 59], subsequent relaxations explored the possibility of local scheduling at each node based on communication events to achieve similar goals [3, 46, 36, 47, 60]. 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 virtualized hosting platforms 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 and accounting mechanisms attempt to address these multiple goals.

Accounting in operating systems and VMMs: Existing research has proposed new OS abstractions and accounting techniques to replace a process as a resource principal [5, 27]. Although our accounting has similar goals, there are some key differences. Our granularity for accounting is a domain, and hence much coarser than a resource container/virtual service. As a result, we need not define a new systems abstraction or present a new API to the programmer as is done by resource containers. Finally, our accounting technique relies on simple page-flip counts due to all I/O virtualization in Xen relying on a common way of using the page-flip mechanism.

In the context of the Xen VMM, two pieces of research are closely related to our work. The first consists of an empirical study of the CPU usage of *Domain0* induced by varying intensities of I/O activities performed by guest domains [15] and a recent paper that describes a technique to account this CPU activity to individual domains [22]. The latter paper measures work done by *driver domains*, privileged domains that implement device drivers rather than *Domain0* implementing them, on behalf of guest domains and feeds this information to a modified SEDF scheduler which maintains a “debt” for guest domains and spreads out the repayment of this debt across several time-periods. It also implements a control mechanism for driver domains to limit network traffic and hence enforce CPU allocation for a guest domain.

The second set of ideas that we would like to point out consists of suggestions that we found in an online discussion by Ian Pratt, one of the developers of Xen [57]. This discussion proposes to maintain counters in the Xen hypervisor to reflect I/O activity done for guest domains, similar to our approach, and using this information to augment the CPU scheduler for preventing I/O-intensive domains from degrading the CPU received by other domains.

Integrated resource management: Our scheduling algorithm is an example of a resource manager that integrates the management of CPU capacity with the usage pattern of another resource, namely, network bandwidth. Resource managers that employ such co-operation with their counterparts for other resources have been developed in other context. Prior work on gang scheduling and co-scheduling are examples of integrated management of CPU and network bandwidth. Similarly, research has been conducted on designing memory managers that co-operate with the CPU scheduler (for example, to help it achieve fair CPU allocations even under memory pressure) both

in the context of traditional operating systems [10] and VMMs [54].

7 Conclusions

Recent advances in software and architectural support for server virtualization have created interest in using this technology to design cluster-based hosting platforms. In particular, there is a lot of interest among providers of Internet data centers as well as administrators of large-scale enterprise applications to exploit features of VMMs such as agile migration and secure co-location of applications for cost-cutting via improved consolidation. We identified two shortcomings in existing VMMs that prove to be obstacles in the efficient operation of such highly consolidated virtualization-based hosting platforms (VHPs): (i) CPU schedulers that are agnostic of the communication behavior of modern, multi-tier applications and (ii) inadequate or inaccurate accounting of the CPU overheads of I/O virtualization. We developed a new communication-aware CPU scheduling algorithm and a CPU usage accounting mechanism to address these problems. We implemented our algorithms in the Xen VMM and built a prototype VHP on a cluster of 36 servers. 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 in the range 20%-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.

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] J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S. Lueng, M. Vandervoorde, C. Waldspurger, and W. Weihl. Continuous Profiling: Where Have All the Cycles Gone? In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 1–14, October 1997.
- [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 Virtualization. In *Proceedings of the Nineteenth SOSP*, 2003.

- [7] R. Barr, Z. J. Haas, and R. van Renesse. JiST: An Efficient Approach to Simulation Using Virtual Machines: Research Articles. *Softw. Pract. Exper.*, 35(6):539–576, 2005.
- [8] 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.
- [9] M. Bennani and D. Menasce. Autonomic Virtualized Environments. In *Proceedings of the IEEE International Conference on Autonomic and Autonomous Systems (ICAS 2006)*, Santa Clara, CA, July 2006.
- [10] E. Berger, S. Kaplan, B. Urgaonkar, P. Sharma, A. Chandra, and P. Shenoy. Scheduler-aware Virtual Memory Management. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP 2003)*, Lake George, NY, October 2003.
- [11] B. Buck and P. Keleher. Locality and Performance of Page- and Object-Based DSMs. In *Proc. of the First Merged Symp. IPPS/SPDP 1998*), pages 687–693, 1998.
- [12] 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.
- [13] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing Server Energy and Operational Costs in Hosting Centers. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2005)*, Banff, Canada, June 2005.
- [14] 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 *Postthe ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2006)*, June 2006.
- [15] L. Cherkasova and R. Gardner. Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor. In *Proceedings of the 2005 USENIX Annual Technical Conference*, pages 387–390, April, 2005.
- [16] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J.N. Matthews. Xen and the Art of Repeated Research. In *Proceedings of FREENIX 2004*, July 2004.
- [17] 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.
- [18] 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.

- [19] 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.
- [20] R. Goldberg. Survey of Virtual Machine Research. *IEEE Computer*, pages 34–45, June 1974.
- [21] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum. Cellular Disco: Resource Management using Virtual Clusters on Shared-memory Multiprocessors. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'99)*, pages 154–169, December 1999.
- [22] 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.
- [23] D. Gupta, R. Gardner, and L. Cherkasova. XenMon: QoS Monitoring and Performance Profiling Tool. Technical Report HPL-2005-187, HP Labs, 2005.
- [24] Intel VT. <http://www.intel.com/technology/itj/2006/v10i3/foreword.htm>.
- [25] The JBoss Application Server. <http://www.jboss.org>.
- [26] 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.
- [27] J.Reumann, A. Mehra, K. Shin, and D. Kandlur. Virtual Services: A New Abstraction for Server Consolidation. In *Proceedings of USENIX Annual Technical Conference*, June 2000.
- [28] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. In *IEEE Journal on Selected Areas in Communication*, 14(7), pages 1280–1297, September 1996.
- [29] 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.
- [30] C. Lin, H. Chu, and K. Nahrstedt. A Soft- Real-time Scheduling Server on the Windows NT. In *Proceedings of the Second USENIX Windows NT Symposium*, Seattle, WA, August 1998.
- [31] 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.
- [32] B. Lim M. Nelson and G. Hutchins. Fast Transparent Migration for Virtual Machines. In *Proceedings of the 2005 USENIX Annual Technical Conference*, pages 391–394, April, 2005.

- [33] 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.
- [34] J. Moore, D. Irwin, L. Grit, S. Sprenkle, and J. Chase. Managing Mixed-Use Clusters with Cluster-on-Demand. Technical report, Department of Computer Science, Duke University, November 2002.
- [35] MySQL. <http://www.mysql.com>.
- [36] 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.
- [37] J. Nieh and M. Lam. A SMART Scheduler for Multimedia Applications. *ACM Transactions on Computer Systems*, 21(2):117–163, 2003.
- [38] 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.
- [39] A. Parekh and R. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks – The Single Node Case. In *Proceedings of IEEE INFOCOM '92*, pages 915–924, May 1992.
- [40] P. Pradhan, R. Tewari, S. Sahu, A. Chandra, and P. Shenoy. An Observation-based Approach Towards Self-Managing Web Servers. In *Proceedings of the Tenth International Workshop on Quality of Service (IWQoS 2002)*, May 2002.
- [41] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. Griffin, and L. van Doorn. Building a MAC-Based Security Architecture for the Xen OpenSource Hypervisor. In *Proceedings of the 2005 Annual Computer Security Applications Conference (ACSAC)*, pages 249–258, 2005.
- [42] 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.
- [43] 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.
- [44] 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.
- [45] J. E. Smith and R. Nair. *Virtual Machines: Architectures, Implementations and Applications*. Morgan Kaufmann, New York, 2004.

- [46] 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.
- [47] 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.
- [48] NYU TPC-W. <http://www.cs.nyu.edu/pdsg/>.
- [49] 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*.
- [50] B. Urgaonkar, A. Rosenberg, and P. Shenoy. Application Placement on a Cluster of Servers. In *Proceedings of the Seventeenth International Conference on Parallel and Distributed Computing Systems PDCS-2004, San Fransisco, CA, September 2004*.
- [51] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic Provisioning of Multi-tier Internet Applications. In *Proceedings of the Second IEEE International Conference on Autonomic Computing (ICAC-05), Seattle, WA, June 2005*.
- [52] 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*.
- [53] VMware. <http://www.vmware.com/>.
- [54] 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.
- [55] 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.
- [56] 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.
- [57] Xen Roadmap, 2006. <http://wiki.xensource.com/xenwiki/XenRoadMap>.
- [58] XenSource: Enterprise-Grade Open Source Virtualization. <http://www.xensource.com/>.

- [59] 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.
- [60] 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.