

Scheduling I/O in Virtual Machine Monitors

Diego Ongaro Alan L. Cox Scott Rixner

Rice University
Houston, Texas, USA
{diego.ongaro,alc,rixner}@rice.edu

Abstract

This paper explores the relationship between domain scheduling in a virtual machine monitor (VMM) and I/O performance. Traditionally, VMM schedulers have focused on fairly sharing the processor resources among domains while leaving the scheduling of I/O resources as a secondary concern. However, this can result in poor and/or unpredictable application performance, making virtualization less desirable for applications that require efficient and consistent I/O behavior.

This paper is the first to study the impact of the VMM scheduler on performance using multiple guest domains concurrently running different types of applications. In particular, different combinations of processor-intensive, bandwidth-intensive, and latency-sensitive applications are run concurrently to quantify the impacts of different scheduler configurations on processor and I/O performance. These applications are evaluated on 11 different scheduler configurations within the Xen VMM. These configurations include a variety of scheduler extensions aimed at improving I/O performance. This cross product of scheduler configurations and application types offers insight into the key problems in VMM scheduling for I/O and motivates future innovation in this area.

Categories and Subject Descriptors C.4.1 [OPERATING SYSTEMS]: Process Management—Scheduling

General Terms Experimentation, Performance

Keywords Machine Virtualization, Network I/O, Scheduling Policy, Server Consolidation, Xen

1. Introduction

In many organizations, the economics of supporting a growing number of Internet-based application services has created a demand for server consolidation. Consequently, there has been a resurgence of interest in machine virtualization [2, 3, 4, 7, 9, 12, 13, 18, 19]. A virtual machine monitor (VMM) enables multiple virtual machines, each encapsulating one or more services, to share the same physical machine safely and fairly. Specifically, it provides

This work was supported in part by the National Science Foundation under Grant Nos. CCF-0546140 and CNS-0720878 and by gifts from Advanced Micro Devices and Hewlett-Packard.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

VEE'08, March 5–7, 2008, Seattle, Washington, USA.
Copyright © 2008 ACM 978-1-59593-796-4/08/03...\$5.00

isolation between the virtual machines and manages their access to hardware resources.

The scheduler within the VMM plays a key role in determining the overall fairness and performance characteristics of the virtualized system. Traditionally, VMM schedulers have focused on fairly sharing the processor resources among domains while leaving the scheduling of I/O resources as a secondary concern. However, this can cause poor and/or unpredictable I/O performance, making virtualization less desirable for applications whose performance is critically dependent on I/O latency or bandwidth.

This paper explores the relationship between domain scheduling in a VMM and I/O performance. For concreteness, the Xen VMM [4, 9] is used with both its current default Credit scheduler and its earlier SEDF scheduler. In addition, this paper examines a number of new and existing extensions to Xen's Credit scheduler targeted at improving I/O performance.

While other studies have examined the impact of VMM scheduling on I/O performance [5, 6, 11], this paper is the first to do so with multiple guest domains concurrently running different types of applications. In particular, processor-intensive, bandwidth-intensive, and latency-sensitive applications are run concurrently with each other to quantify the impacts of different scheduler configurations on processor and I/O performance. This cross product of scheduler configurations and application types offers several insights into the key problems in VMM scheduling for I/O and motivates future innovation in this area.

Both the Credit and SEDF schedulers within Xen do a good job of fairly sharing processor resources among compute-intensive domains. However, when bandwidth-intensive and latency-sensitive domains are introduced, the schedulers achieve mixed results, depending on the particular configuration.

The default scheduler in the current version of Xen is a credit scheduler which uses a credit/debit system to fairly share processor resources. The Xen Credit scheduler is invoked whenever an I/O event is sent and boosts the priority of an idle domain receiving an I/O event. It does not, however, sort domains in the run queue based on their remaining credits. This study shows that these extensions can have significant effects on I/O performance. In particular, the priority boost optimization frequently impacts both bandwidth and latency positively, as it allows domains performing I/O operations to achieve lower response latency. Sorting the run queue based on remaining credits can have a similarly positive effect, as it allows infrequently running, but latency-sensitive, domains to run more promptly. When events are sent, tickling the scheduler to immediately preempt a lower priority domain can sometimes reduce response latency. However, it also creates a wide variance in both bandwidth and latency among I/O-intensive domains. Finally, this study shows that latency-sensitive applications will perform best if they are not combined in the same domain with a compute-intensive application, but instead are placed within their own domain.

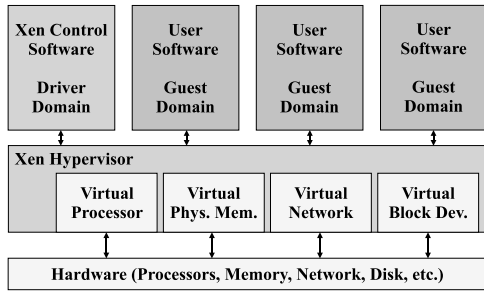


Figure 1. The Xen Virtual Machine Environment

The rest of this paper is organized as follows. The next section provides a brief introduction to Xen. Section 3 describes Xen’s default scheduler, the Credit scheduler. Section 4 discusses modifications to the Credit scheduler as well as the earlier SEDF scheduler. Section 5 describes the experimental methodology and Section 6 discusses the experimental results. Section 7 discusses the related work. Finally, Section 8 concludes the paper.

2. The Xen VMM

A virtual machine monitor allows multiple operating systems to share a single machine safely. It provides isolation between operating systems and manages access to hardware resources.

Figure 1 graphically shows the organization of Xen, an open source virtual machine monitor based on the Linux kernel [4]. Xen effectively consists of two elements: the hypervisor and the driver domain. The hypervisor provides an abstraction layer between the guest operating systems running in their own domains and the actual hardware. In particular, this layer performs functions such as scheduling processors and allocating memory among guest domains. One of the major functions of the driver domain is to provide access to the actual hardware I/O devices. The hypervisor grants the driver domain direct access to the devices and does not allow the guest domains to access them directly. Therefore, all I/O traffic must pass through the driver domain.

In Xen, the hypervisor therefore protects the guest domains from each other and shares I/O resources through the driver domain. This enables each guest operating system to behave as if it were running directly on the hardware without worrying about protection and fairness.

3. Xen’s Credit Scheduler

3.1 Operation

The current version of the Xen virtual machine monitor (version 3) uses a credit scheduler to schedule domains. Each domain is given a certain number of *credits* by the system administrator. The overall objective of the Credit scheduler is to allocate the processor resources fairly, weighted by the number of credits each domain is allocated. Therefore, if each domain is given the same number of credits, the domains should expect an equal fraction of processor resources.

In practice, the Credit scheduler works as follows. Domains can be in one of two states: *OVER* or *UNDER*. If they are in the *UNDER* state, then they have credits remaining. If they are in the *OVER* state, then they have gone over their credit allocation. Credits are debited on periodic scheduler interrupts that occur every 10ms. At each scheduler interrupt, the currently running domain is debited 100 credits. When the sum of the credits for all of the domains in the system goes negative, all domains are given new credits. When making scheduling decisions, domains in the *UNDER* state are always run before domains in the *OVER* state. A domain that is

over its credit allocation is only executed if there are no domains in the *UNDER* state that are ready to run. This allows domains to use more than their fair share of the processor resources only if the processor(s) would otherwise have been idle.

When making scheduling decisions, the Credit scheduler only considers whether a domain is in the *OVER* or *UNDER* state. The absolute number of credits that a domain has remaining is irrelevant. Rather, domains in the same state are simply run in a first-in, first-out manner. Domains are always inserted into the run queue after all other domains in the run queue that are in the same state, and the scheduler always selects the domain at the head of the run queue to execute. When a domain reaches the head of the run queue, it is allowed to run for three scheduling intervals (for a total of 30ms) as long as it has sufficient credits to do so.

3.2 Events

Xen uses *event channels* to communicate virtual interrupts. The hypervisor sends an event to the driver domain in response to a physical interrupt. The next time the driver domain runs, it will see the event and process the interrupt. Events are also used for inter-domain communication. After a domain sends an event to another domain, the receiving domain will see and process the event the next time it runs.

For example, when the network interface card (NIC) receives a network packet, it generates an interrupt. The hypervisor forwards that IRQ to the driver domain as an event. When the driver domain runs, it sees the event. If the packet is destined for a guest domain, the driver domain then copies the packet to the appropriate guest domain and sends that domain an event. Once the guest domain runs, it sees the event and handles the packet.

If the guest domain responds to the packet, it remaps the response packet(s) to the driver domain and sends an event to the driver domain. When the driver domain runs, it sees the event and responds by forwarding the packet to the NIC. The NIC’s acknowledgement (another hardware interrupt) is forwarded by the hypervisor to the driver domain as an event. The driver domain then sends an event to the guest domain so that it will know that the packet has been sent.

Whenever an event is sent to a domain, the hypervisor wakes the target domain if it is idle. Then, the hypervisor *tickles* the scheduler, meaning that it invokes the scheduler to re-evaluate which domain should be running. If the woken domain has a higher priority than the currently running domain, the scheduler will switch to the newly woken domain. Tickling the scheduler in this manner potentially reduces the communication latency for both hardware interrupts and inter-domain notifications by immediately running the domain receiving the event.

3.3 Fairness

The Credit scheduler attempts to fairly share the processor resources. Figure 2 shows the fraction of the processor that is allocated among seven domains with equal credit allocations that are running concurrently. In this experiment, each domain is simply burning as many processor cycles as it can in an infinite loop. As the figure shows, the Credit scheduler is quite successful at fairly sharing the processor in this case. Furthermore, modifying the domains’ credit allocation should directly affect the fraction of time each domain receives in an obvious way.

However, the scheduler is fairly sharing the processor resources only by approximation. For example, a domain that runs for less than 10ms will not have any credits debited. When all domains are primarily using processor resources, this is unlikely to have a significant impact on the fair sharing of the processor resources. Recent activity in the Linux community, however, has focused on replacing this type of approximate scheduler with a “completely

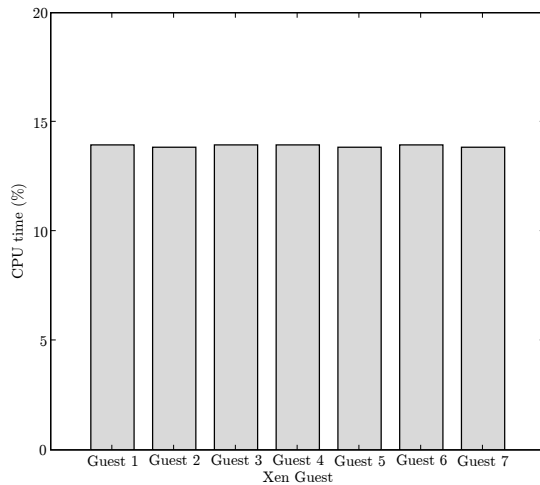


Figure 2. CPU fairness with Xen’s default configuration.

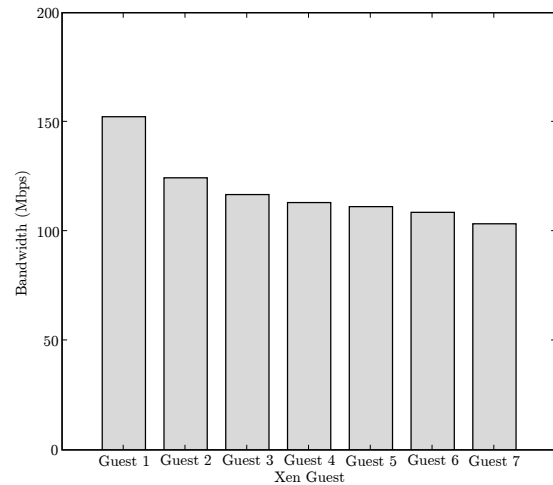


Figure 3. I/O fairness with Xen’s default configuration.

fair scheduler,” which keeps track of the time each process uses with nanosecond granularity. Such a scheduler can more fairly allocate processor resources than a statistical sampling scheduler like the Credit scheduler.

Regardless of the accounting method used in the scheduler, the driver domain is treated just like any other domain. While it can be assigned more credits than other domains, allowing it to consume a larger fraction of the processor resources, it receives no additional priority and is still scheduled based on its state and the order it is inserted into the run queue. When low response latency from the driver domain is not needed, this is entirely appropriate. However, if low response latency is desired—as in a networking application, for example—then this policy may not be effective.

Since the Credit scheduler does not consider *when* each domain should receive its allocated fraction of processor resources, it does not provide fair latency. For example, a domain that performs a lot of I/O will be treated similarly to a domain that only performs processing. Although an I/O-intensive domain may actually consume far less processor resources than the processing domain, it may still perform poorly. This occurs because the processing domain can consume 30ms of processing time once it arrives at the head of the run queue, regardless of whether there are pending I/O events. If the I/O events can be handled quickly, the domain performing I/O would benefit from being prioritized above the processing domain. This would allow that domain to promptly handle the I/O operation, without much harm to the processing domain.

Fundamentally, the Credit scheduler has no notion of the *urgency* with which a domain needs to execute. Rather, it only attempts to fairly allocate processor resources among domains over the long run. This results in a second fundamental problem for I/O, which is that response latency will vary widely among domains. Ultimately, the latency between the occurrence of an I/O event and the scheduling of the appropriate domain is determined by the position of that domain in the run queue. If the domain is close to the head of the queue, its response latency will be lower. If it is far from the head of the queue, behind a lot of compute-intensive domains with many remaining credits, its response latency can be quite high. This leads to the unexpected situation where different domains experience widely varying I/O performance, despite being allocated an equal number of credits and seemingly having equal priorities.

Figure 3 further illustrates this problem. The figure shows the network bandwidth achieved by seven domains that are running concurrently. Each domain is running a network streaming benchmark, which attempts to send a continuous stream of data as fast as possible. In contrast to processor-intensive workloads, it is obvious that the Credit scheduler does not achieve fairness for I/O-intensive workloads. The bandwidth achieved by the domains varies by a factor of almost 1.5. This is a direct result of the uneven latency experienced by the domains. As TCP throttles bandwidth based on latency, each domain stabilizes at a different level of bandwidth based on the idiosyncrasies of the scheduler.

These two phenomena, high and uneven I/O response latency, make I/O-intensive domains perform poorly and unpredictably. This limits the usefulness of virtualization for many types of server applications that rely heavily on networking and other I/O.

3.4 Boosting I/O Domains

In an attempt to solve the problem of high response latency, the Credit scheduler adds an additional state: `BOOST`. A domain in the `BOOST` state has a higher priority in the scheduler than domains in the `UNDER` and `OVER` states. The `BOOST` state is meant to provide a mechanism for domains to achieve low I/O response latency.

A domain enters the `BOOST` state when it receives an event over an event channel while it is idle. This prevents the domain from entering the run queue at the tail and having to wait for all other active domains before being executed. Since the scheduler is tickled when an event is sent, this means that if the domain receiving the event is boosted, it will very likely preempt the current domain and begin running immediately.

Increasing a domain’s priority in this fashion can have some effect on lowering the high latency experienced by domains performing I/O, as shown in Figure 4. The figure shows the ping latency from an external machine to an otherwise idle guest domain with and without boosting. There are seven other domains consuming processor cycles while the test is running. The graph shows the ping latency for 90 consecutive pings. The solid line shows that with boost disabled, the latency oscillates around 150ms. The high latency is the result of the ping domain having to wait behind the processing domains for service. The dotted line shows that when boost is enabled, the ping latency is usually around 0.2–0.3ms, with

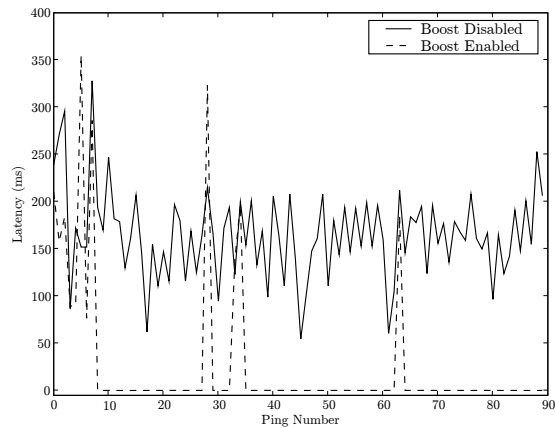


Figure 4. The effect of boost on latency in the Credit scheduler. One guest is receiving pings, while 7 others are performing processor-intensive activities. The event channel fix (described in section 4.1) is active.

some spikes when the scheduler is occasionally unable to give the ping domain prompt service.

While boosting reduces response latency, it does very little to improve the uneven I/O response latency when multiple domains are performing I/O. In that case, many domains get boosted, which negates the value of the BOOST state. For example, boost has little effect on the experiment shown in Figure 3. With boosting enabled or disabled, the variation in achieved bandwidth is nearly identical.

4. Scheduler Enhancements for I/O

This section first discusses a change to Xen’s event channel mechanism that results in fairer scheduling of I/O-intensive domains. Then, it discusses two modifications to the Credit scheduler that are intended to improve I/O performance. Finally, it describes an alternative to the Credit scheduler, Xen’s SEDF scheduler.

4.1 Fixing Event Channel Notification

Xen uses event channels, as described previously, to communicate IRQ’s to the driver domain. A two-level hierarchy of bit vectors, referred to as the pending array, is shared between the hypervisor and the driver domain’s kernel. Each bit in the second level of the hierarchy corresponds to a *port*, which maps to an IRQ number in this case. A set bit at this level indicates the port has a pending event. The first level of the hierarchy is used as an optimization to avoid excessive searching for set bits. A bit is set in the first level if any bit is set in the corresponding vector in the second level.

When a port number is allocated, Xen simply uses the smallest port number not in use. Thus, since physical devices require port numbers immediately when the driver domain boots, they are assigned relatively low port numbers. Any virtual devices to support guest operations, such as loopback filesystems or virtual network interfaces, will be assigned higher port numbers. Similarly, guest domains will be assigned port numbers in the order that they are started.

The event channel driver, which runs in the driver domain, scans the two-level hierarchy of bit vectors sequentially. When it finds a bit set in the first level of the hierarchy, it scans for set bits in the corresponding bit vector in the second level of the hierarchy. When it finds a set bit, it trivially calculates the port number from the index into the first and second levels of the hierarchy and maps

that port into an IRQ number. It then calls the appropriate handler for that IRQ. After processing the IRQ, it restarts the scan from the beginning of the current bit vector in the second level of the hierarchy.

There is a subtle, but rather important, issue with the event channel driver. Suppose that Guest 4 is receiving a continuous stream of data. When a packet for Guest 4 arrives over the network, the driver domain receives an interrupt from the NIC through the event channel mechanism. It processes the IRQ, finds the packet’s destination is Guest 4, copies the packet to Guest 4, and finally sends an event channel notification to Guest 4. Xen wakes Guest 4 and tickles the scheduler. The Credit scheduler may then preempt the driver domain in favor of Guest 4. Guest 4 may send an acknowledgement packet back to the driver domain followed by an event channel notification. When the driver domain is next executed, it has a pending event from Guest 4. Also, the NIC has likely received more inbound data destined for Guest 4, so the driver domain has the event bit set for the port corresponding to the NIC’s IRQ. Since the driver domain always seeks the first set bit in the bit vector, it will always process lower port numbers before higher port numbers. Moreover, because of the way port numbers are allocated, inbound packets destined for Guest 4 will always be processed before outbound packets originating from Guest 4. If Xen repeatedly preempts the driver domain to inform Guest 4 of new inbound data, the system enters a cycle where it becomes unlikely that the driver domain will process any outbound acknowledgements.

The event channel driver was modified to solve this problem in a straight-forward manner. Rather than restarting the scan from the beginning of the bit vector after processing an IRQ, the scan resumes from where it left off. Once the end of the two-level hierarchy is reached, the scan will restart from the beginning. This guarantees that no port will be processed a second time before all other pending ports have been processed once.

This fix was submitted to Xen’s maintainers and is now incorporated in Xen’s development code branch (changeset 324 of the linux-2.6.18-xen.hg repository).

4.2 Minimizing Preemptions

Under Xen’s network I/O architecture, arriving packets are first delivered to the driver domain. The driver domain demultiplexes the arriving packets, forwarding each one to the appropriate guest domain. As described in Section 3.2, the driver domain uses an event channel to signal the availability of one or more packets to a guest domain. If that guest domain was idle, it is woken as a result.

When multiple packets destined for distinct domains are delivered to the driver domain, they are processed in an arbitrary order. The driver domain sends an event to the appropriate guest domain as it forwards each packet. Since the act of sending an event tickles the scheduler, the scheduler may preempt the driver domain before it has processed all of the packets. This can have an effect that is similar to *priority inversion*. For example, consider a scenario in which two packets arrive: the first is destined for a guest domain that is UNDER, and the second is destined for a guest domain that is or will be BOOST. If the driver domain is OVER, it will be preempted after the first packet is forwarded, thereby delaying the delivery of the second packet.

Arguably, the driver domain should not be preempted while it is demultiplexing packets. It is a trusted component in Xen’s I/O architecture, and thus can be expected to be well-behaved in its use of computational resources. To avoid preempting the driver domain, tickling was disabled altogether. This forces the scheduler to wait for the driver domain to yield before it can run any other guest.

4.3 Ordering the Run Queue

Under the Credit scheduler, I/O-intensive domains will often consume their credits more slowly than CPU-intensive domains. In fact, an I/O-intensive domain will not be debited any credits if it happens to block before the periodic scheduler interrupt, which occurs every 10ms. However, when it later becomes runnable, its remaining credits have only a limited effect on its place in the run queue. Specifically, as described in Section 3.1, the number of remaining credits only determines the domain’s state. The domain is always enqueued after the last domain in the same state. Intuitively, sorting the run queue by the number of remaining credits that each domain possesses could reduce the latency for an I/O-intensive domain to be executed. The Credit scheduler was modified to insert domains into the run queue based on their remaining credits in order to evaluate this optimization.

4.4 Scheduling Based on Deadlines

In principle, under a workload mixing CPU- and I/O-intensive domains, Xen’s *Simple Earliest Deadline First* (SEDF) scheduler should allow I/O-intensive domains to achieve lower latency.

With the SEDF scheduler, each domain is allocated processing resources according to two parameters: the domain’s *period* and the domain’s *slice*. In particular, the SEDF scheduler guarantees that if the domain is runnable and not blocked, it will be executed for the amount of time given by its slice during every interval of time given by its period. Thus, the administrator can determine the interval over which fair processor resource allocation is achieved by proportionally changing the period and slice.

The SEDF scheduler operates as follows. It maintains for each domain a *deadline*, the time at which the domain’s current period ends, and the amount of processing time that the domain is due before the deadline passes. It orders the domains in the run queue according to their deadlines and executes the domain with the earliest deadline. As domains consume processing time, their deadlines will move forward in time. I/O-intensive domains that consume little processing time will typically have earlier deadlines, and thus higher priority, than CPU-intensive domains.

Despite the SEDF scheduler’s advantages for workloads that combine CPU- and I/O-intensive domains, the Credit scheduler replaced the SEDF scheduler as the default scheduler in recent versions of Xen. The reasons given for this change were that the Credit scheduler improves scheduling on multiprocessors and provides better QoS controls [1].

5. Experimental Methodology

5.1 Benchmarks

Three simple microbenchmarks were used to characterize the behavior of the VMM scheduler across workloads:

- **burn**: This microbenchmark attempts to fully utilize a guest domain’s processor resources. The burn script simply runs an infinite computation loop within the guest. This script will consume as many processor resources as the VMM will allow.
- **stream**: This microbenchmark attempts to fully utilize a guest’s network resources. A remote system is used to stream as much data as possible over the network to the guest. This benchmark will consume as many network resources as the VMM will allow.
- **ping**: This microbenchmark attempts to achieve low response latency. A remote system is used to send pings to the guest. The ping program on the remote system transmits each packet upon the receipt of the previous packet’s acknowledgement or after a

Label	Scheduler	Event Ch.	Tickle	Boost	RQ Sort
orig	credit	default	on	on	off
all off	credit	patched	off	off	off
t	credit	patched	on	off	off
b	credit	patched	off	on	off
s	credit	patched	off	off	on
tb	credit	patched	on	on	off
ts	credit	patched	on	off	on
bs	credit	patched	off	on	on
tbs	credit	patched	on	on	on
SEDF	SEDF	default	N/A	N/A	N/A
SEDF'	SEDF	patched	N/A	N/A	N/A

Table 1. Various configurations of the scheduler tested.

timeout. As the guest domain does nothing else, this benchmark will achieve the lowest response latencies the VMM will allow.

These benchmarks are simple representatives of a processor-intensive workload, a network-intensive workload, and a latency-sensitive workload.

5.2 Experiments

The experiments used the following test suite that combines the three microbenchmarks in eight different ways:

1. stream x 7: The first seven guests running stream; the last guest was idle.
2. stream x 3, burn x 3: The first three guests running stream; the next three guests running burn.
3. burn x 7, ping x 1: The first seven guests running burn; the last guest receiving pings.
4. ping x 1, burn x 7: The first guest receiving pings; the last seven guests running burn.
5. stream x 7, ping x 1: The first seven guests running stream; the last guest receiving pings.
6. ping x 1, stream x 7: The first guest receiving pings; the last seven guests running stream.
7. stream x 3, burn x 3, ping x 1: The first three guests running stream; the next three guests running burn; the last guest receiving pings.
8. stream x 3, ping+burn x 1, burn x 3: The first three guests running stream; the next guest receiving pings and running burn; the final three guests only running burn.

The first two tests show the ability of the scheduler to fairly allocate processor resources to bandwidth-intensive I/O domains. Tests 3 through 7 show the effects of the scheduler on the response latency of the domain running ping under various configurations. Finally, test 8 shows whether or not using more processor resources can enable a domain to achieve lower response latency.

5.3 Scheduler Configurations

Table 1 shows the 11 scheduler configurations that were evaluated. The “orig” configuration is the default configuration in Xen 3 unstable. As the table shows, it uses the Credit scheduler, does not include the event channel fix described in Section 4.1, tickles the scheduler when sending events as described in Section 3.2, includes the BOOST state described in Section 3.4, and does not sort the run queue based on remaining credits as described in Section 4.3. The “all off” configuration includes the event channel fix, but otherwise turns off all other modifications to the Credit scheduler. The following 7 configurations are all of the combinations of the tickling (t),

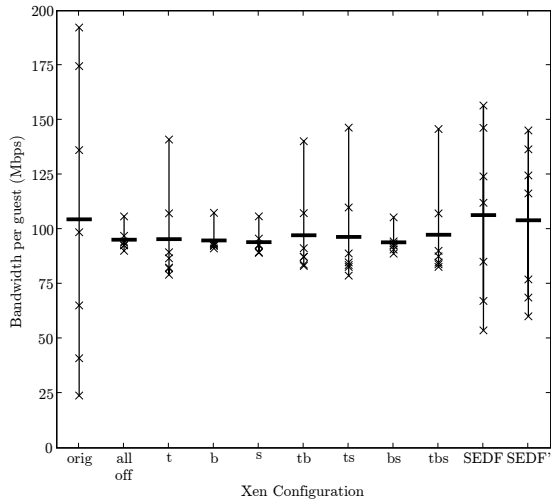


Figure 5. Bandwidth distributions in the stream x 7 test.

boosting (b), and run queue sorting (s) modifications. The final two configurations use the SEDF scheduler, as described in Section 4.4, instead of the Credit scheduler. The “SEDF” configuration does not include the event channel fix, and the “SEDF’” configuration does. Note that the three modifications to the Credit scheduler are not relevant in the SEDF scheduling algorithm.

5.4 Experimental System

All experiments were run using Xen 3 on an AMD Opteron-based system. The system includes an Opteron 242 processor, 2 GB of memory, and two Ethernet network interfaces. A Gigabit Ethernet network interface was used for the `stream` and `ping` tests, while a 100 Mbps Ethernet network interface was used for control messages to setup and manage the tests. An Intel Pentium 4-based system running native Linux served as the other endpoint for the `stream` and `ping` tests. This endpoint system was never the bottleneck in any experiments.

The virtualized system ran Xen 3 unstable (changeset 15080; May 17, 2007). The driver domain and all guest domains ran the Ubuntu Dapper Linux distribution with the Linux 2.6.18 kernel. All of the eight guest domains were identical for all experiments. The guests shared a read-only root filesystem and each had its own read-write `/var` filesystem. A `sysfs` interface in the driver domain was created to toggle the scheduler parameters in the tests.

6. Experimental Results

6.1 Bandwidth (Tests 1–2)

Figure 5 presents the results of the first test: stream x 7. The graph shows the bandwidth achieved by each of the seven guest domains for each of the scheduler configurations described in Section 5.3. The x-axis presents each of the 11 scheduler configurations. The left y-axis shows the bandwidth in Mbps. For a particular configuration, each guest’s bandwidth is marked with an `x`, so there are seven `x`’s on each configuration’s line. The average bandwidth achieved for the configuration is shown as a thick horizontal line. All bandwidth results throughout this section will be plotted in this manner, using the left y-axis to show the achieved bandwidth for all guests running the stream benchmark.

As the figure shows, while the original Xen scheduler achieves nearly the best average bandwidth, the spread in bandwidth among

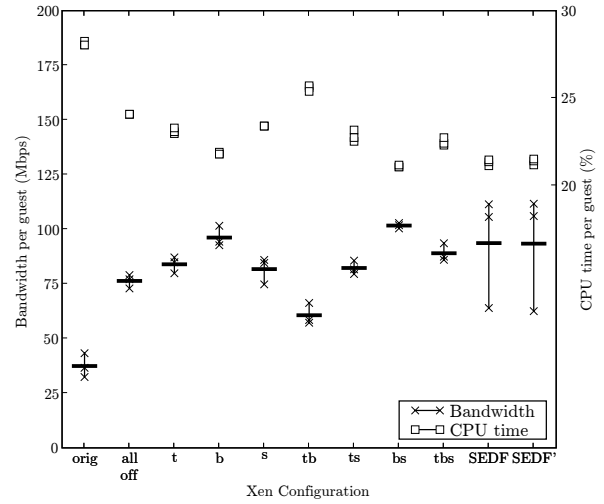


Figure 6. Bandwidth and CPU distributions in the stream x 3, burn x 3 test.

the guests is the worst. The lowest bandwidth domain achieved only 23.9 Mbps, whereas the highest bandwidth domain achieved 192.3 Mbps with the remaining guests relatively evenly spaced between. The wide variance is a direct result of the event channel code problems that were discussed in Section 4.1. With this code fixed and everything else the same (configuration “tb” in the figure), the variance in bandwidth is much lower, but it is still significant. The figure identifies the scheduler tickling that occurs as a result of event channel notifications to be the culprit of the unfairness in the Credit scheduler once the event channel code is fixed. In fact, all configurations that include tickling (t) have a reasonably wide bandwidth distribution, whereas those configurations that don’t keep the bandwidth differences between domains below 17 Mbps.

While the SEDF scheduler achieves a slightly higher average bandwidth and a smaller range than the original Credit scheduler, it still does not effectively balance the I/O use of the domains. In fact, the bandwidth variation among domains with the SEDF scheduler is much higher than with the Credit scheduler when the event channel code is fixed.

Figure 6 presents the results of the second test: stream x 3, burn x 3. As before, the range of bandwidths for the three stream domains is plotted against the left y-axis. This figure also introduces the right y-axis, which shows the processor utilization of the burn domains. For a particular configuration, each guest’s processor utilization (as a percentage of the total processor resources) is marked with a box, so there are three boxes on each configuration’s line. In this test, as with most tests, the variations in processor utilization are not visible as all the scheduler configurations are able to fairly allocate processor resources among the burn domains. All processor utilization results throughout this section will be plotted in this manner, using the right y-axis to show processor utilization for all guests running the burn benchmark.

This figure further highlights the problems with fairness for the bandwidth domains. The original Xen configuration performs the worst of all of the configurations, allocating the most processor resources to the burn domains and severely limiting the achieved bandwidths of the stream domains. The variation in bandwidth among the stream domains, however, is reduced compared to the stream x 7 test.

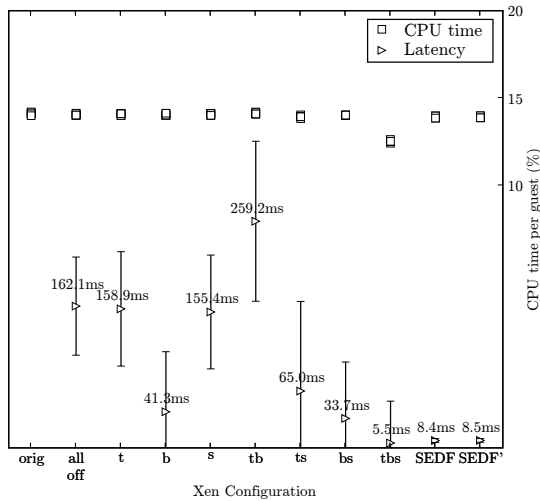


Figure 7. CPU distributions with latency in the burn x 7, ping x 1 test. The latency for the original case (beyond the scale shown here) is 36.6 seconds.

This figure clearly shows the value of the boost (b) optimization. The processor resources allocated to the burn domains are fairly constant across all configurations, but those that include the boost optimization yield the largest average network bandwidths. The best configuration also includes a sorted run queue (s), which yields over 100 Mbps on average with very little variation among domains. For this test, the scheduler tickling optimization not only seems to increase the bandwidth variation, but also lowers the average achievable bandwidth. Again, the SEDF scheduler achieves high average bandwidth with large bandwidth variations.

These two tests clearly show that tickling the scheduler when events are sent to domains creates unfair variations in achievable bandwidth, while the boost optimization helps prevent compute-intensive domains from starving domains performing I/O. In either case, despite being designed for I/O performance, the SEDF scheduler does a poor job of fairly balancing bandwidth across domains.

6.2 Response Latency (Tests 3–7)

Figures 7 and 8 present the results of the third and fourth tests: seven domains running burn and one domain running ping. The only difference is whether the ping domain is booted last (Figure 7) or first (Figure 8). In these figures, processor utilization for the burn domains are plotted as boxes against the right y-axis. These figures introduce ping latency for the ping domain plotted as a triangle with error bars. The triangle is the average latency and is annotated with the actual value. The error bars indicate the standard deviation of the ping latency over the duration of the test. The latency is not plotted against either the left y-axis (reserved for bandwidth) or the right y-axis (reserved for processor utilization). All ping latency results will be reported in this manner throughout the section.

As the ping domain uses almost no processor resources, both figures show that, for most configurations, all seven of the domains running the burn benchmark consistently consume approximately one seventh of the processor resources, as expected. The configuration with all of the Credit scheduler optimizations applied is the exception and will be discussed below. Therefore, the variations lie mostly in the achieved ping latency. Furthermore, the ping latency is so high under the original Xen configuration in both cases that it does not fit on the graph. The ping latency in the original configura-

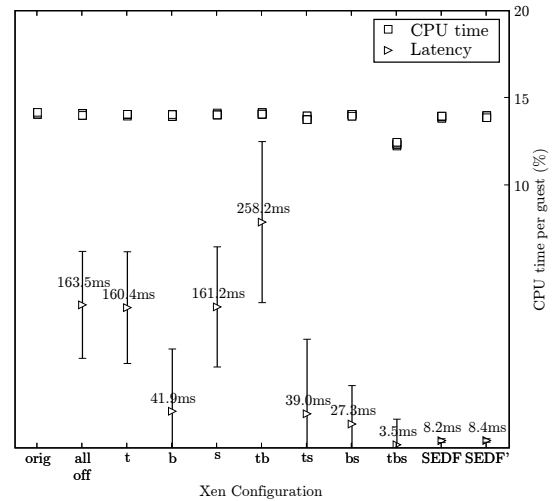


Figure 8. CPU distributions with latency in the ping x 1, burn x 7 test. The latency for the original case (beyond the scale shown here) is 12.3 seconds.

tion is 36.6 seconds for the burn x 7, ping x 1 test and 12.3 seconds for the ping x 1, burn x 7 test. This is due to the misbehaving event channel code in Xen that was fixed for the other configurations, as discussed in Section 4.1.

As Figure 7 shows, when the ping domain is booted last, ping latencies vary from 5.5 ms to 259 ms. Despite having slightly higher latency than the best performing Credit scheduler configuration, the SEDF configurations perform best, as they give much higher priority to the domain running ping. This allows the ping domain to preempt the domains running burn when it needs to, resulting in low latency with very little variation. For the Credit scheduler to compete with the SEDF scheduler, all of the optimizations (tickling (t), boosting (b), and run queue sorting (s)) are needed. When all optimizations are enabled, the average ping latency is lower under the Credit scheduler than the SEDF scheduler, but with much higher variation. The Credit scheduler also reduces the processor utilization of the guests running the burn benchmark in order to achieve these low latencies. The driver domain consumes the remaining processor resources, which helps provide the reduced latency.

When the domain running ping is booted first, as shown by Figure 8, the performance is roughly the same. However, ping latencies do reduce under a few of the Credit scheduler configurations. When all optimizations to the Credit scheduler are enabled, for instance, the ping latency drops from 5.5ms to 3.5ms. The reduction is a result of the event channel code inherently prioritizing domains based on the order in which they boot.

Figures 9 and 10 present the results of the fifth and sixth tests: seven domains running stream and one domain running ping. The only difference is whether the ping domain is booted last (Figure 9) or first (Figure 10). All of the elements of these graphs have been previously introduced.

In both tests, the performance of the domains running the stream benchmark is basically the same as when there was no domain running the ping benchmark (compare to Figure 5). In the original Xen scheduler, the order in which the domain running the ping benchmark was booted has a dramatic effect on ping response latency, which is 2.6ms when the ping domain is booted first and 67.2ms when the ping domain is booted last. This anomalous behavior is

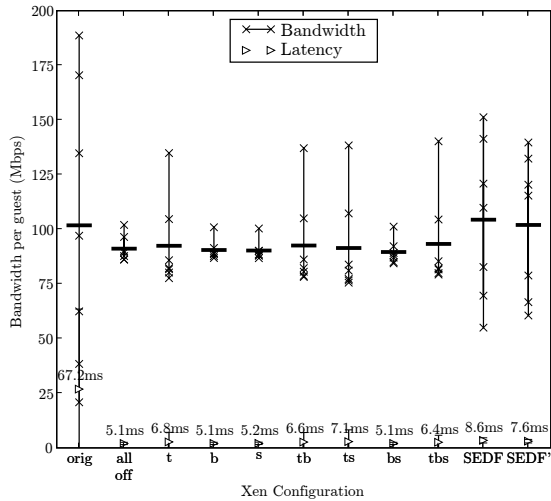


Figure 9. Bandwidth distributions with latency in the stream x 7, ping x 1 test.

due to the broken event channel code in Xen that was fixed for the other configurations.

Even with the fix, the ping latencies are always lower when the domain running the ping benchmark is booted first. This is due to the inherent prioritization of I/O events based on the boot order of domains. While this is arbitrary and perhaps unintended, it does have a noticeable effect. This makes it difficult to draw unambiguous conclusions about the best scheduler configuration for response latency when other domains are bandwidth-intensive. However, as with the bandwidth benchmarks, the scheduler tickling optimization creates wider variations in the ping latency. As long as the event channel fix is employed, the rest of the configurations perform similarly to the “all off” case. This implies that the boosting and run queue sorting optimizations have little effect on response latency when many I/O-intensive domains are active.

Ping latency is higher when using the SEDF scheduler than when using the Credit scheduler for these tests. Furthermore, the variance in bandwidth is also higher when using the SEDF scheduler than when using the Credit scheduler. These differences are consistent across all configurations of the Credit scheduler that include the event channel fix.

Figure 11 presents results for the seventh test, which includes domains running all three types of benchmarks. This is basically the same as test 2, shown in Figure 6, with an additional guest domain running the ping benchmark. The additional ping domain has little impact on the performance of the stream and burn domains, which can be seen by comparing Figures 11 and 6. These latencies are still much lower for most configurations than the achieved latencies when there are burn domains but no stream domains, as in tests 3 and 4, since the ping domain must compete with fewer burn domains for service. The stream domains consume fewer processor resources, so they do not starve the ping domain nearly as much. The variance introduced by the scheduler tickling optimization is much lower than in previous tests, but is still larger than without the optimization. In this test, boosting and run queue sorting appear to be important optimizations that combine to give good performance across all three types of domains.

In this test, the SEDF scheduler configurations enable the ping domain to achieve the lowest ping latency. However, as in previous

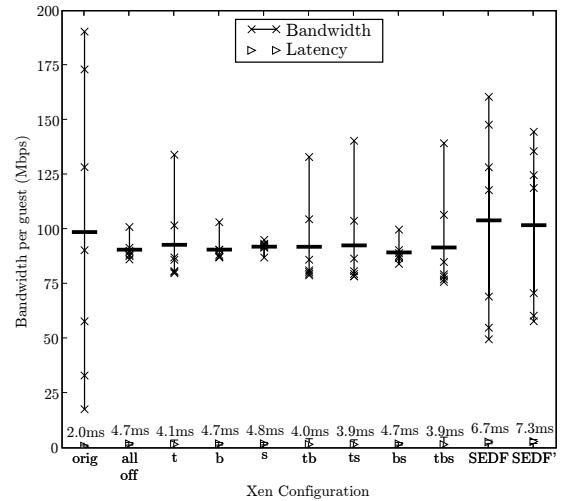


Figure 10. Bandwidth distributions with latency in the ping x 1, stream x 7 test.

tests, the bandwidth variation of the stream domains is larger than most of the Credit scheduler configurations.

These five tests have shown that the boosting and run queue sorting optimizations are important for achieving low response latency. The results with scheduler tickling, however, are much more mixed. When there is only a single domain performing I/O, tickling is extremely effective at reducing latency. However, when multiple domains are competing for I/O resources, tickling creates a large variance in performance. This suggests that it is probably a good compromise not to tickle the scheduler during event delivery. However, the effect of not tickling when there is only a single latency-sensitive domain is large enough on ping latency that there is definitely a need for more work in VMM scheduler design. The SEDF scheduler is not sufficient to solve these problems. While it often reduces response latency, it does so at the cost of large variations in performance for bandwidth-intensive domains.

6.3 Mixed Domains (Test 8)

The test presented in Figure 12 is similar in most respects to that in Figure 11. The difference is that the domain receiving and responding to the ping in Figure 12 is also running burn. The most striking difference between the figures is the significant increase in the response latency for ping in Figure 12. This result can be explained by the fact that the ping-and-burn domain is consuming its credits just as fast as the other burning domains. Therefore, latency optimizations such as boost and run queue sorting will no longer result in preferential scheduling of the ping-and-burn domain.

These results suggest that it is a bad idea to mix compute-intensive and latency-sensitive applications in a single domain within a VMM. The latency-sensitive application will receive better performance if it is running separately in its own domain.

At first, this result seems unsurprising. However, such an increase in ping response time would not occur under native Linux. The reason is that in native Linux, the I/O stack is trusted. Therefore, the OS scheduler can give preferential treatment to I/O operations within the OS, knowing they will perform only the necessary tasks and will complete promptly. In contrast, the I/O stacks within the domains of a VMM cannot be trusted. If the VMM’s scheduler gives preferential treatment to a domain because of a pend-

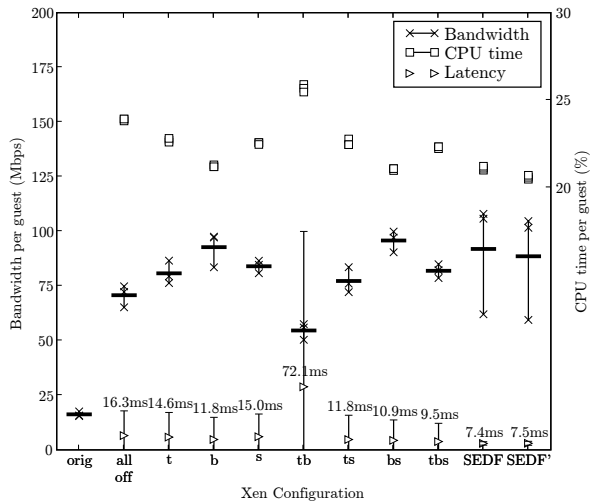


Figure 11. Bandwidth and CPU distributions with latency in the stream x 3, burn x 3, ping x 1 test. The latency for the original case (beyond the scale shown here) is 1.8 seconds, and the average CPU usage per burning domain is 30.5 %.

ing I/O operation, the VMM has no guarantee that the domain will only process that I/O operation and voluntarily return control to the VMM. This makes it difficult for the VMM scheduler to give preferential treatment to I/O operations without sacrificing fairness among the domains.

7. Related Work

Using Xen, Cherkasova *et al.* studied the impact that three different schedulers have on the throughput of three I/O-intensive benchmarks [6]. In addition to the Credit and SEDF schedulers, their study included Xen’s *Borrowed-Virtual-Time* (BVT) scheduler [8]. Their three I/O-intensive benchmarks are a disk read test using the Unix *dd* program, a network bandwidth test, and a web server throughput test. Their study differs from this one in that they evaluate how a single instance of these applications is affected by the choice of scheduling policy. In effect, they evaluate how the scheduler divides the processing resources between the guest domain and the driver domain. In contrast, this paper examines how multiple, concurrent applications are impacted.

Govindan *et al.* propose modifications to the SEDF scheduler [10] that preferentially schedule I/O-intensive domains. Whereas the SEDF scheduler always selects the domain with the earliest deadline, their scheduler takes into consideration the amount of communication being performed by each domain. Specifically, they count the number of packets flowing into or out of each domain and select the domain with the highest count that has not yet consumed its entire slice during the current period. Intuitively, this approach is problematic when there are bandwidth-intensive and latency-sensitive domains running concurrently. The bandwidth-intensive domains are likely to take priority over any latency-sensitive domains with little traffic. For example, in the stream x 7, ping x 1 test presented here, the ping latency is likely to be very high using their approach.

Gupta *et al.* introduce the SEDF-DC scheduler for Xen [11]. The SEDF-DC scheduler is derived from Xen’s SEDF scheduler. It differs from the SEDF scheduler in the respect that it charges guest domains for the time spent in the driver domain on their behalf. In a sense, the SEDF-DC scheduler is addressing the opposite concern

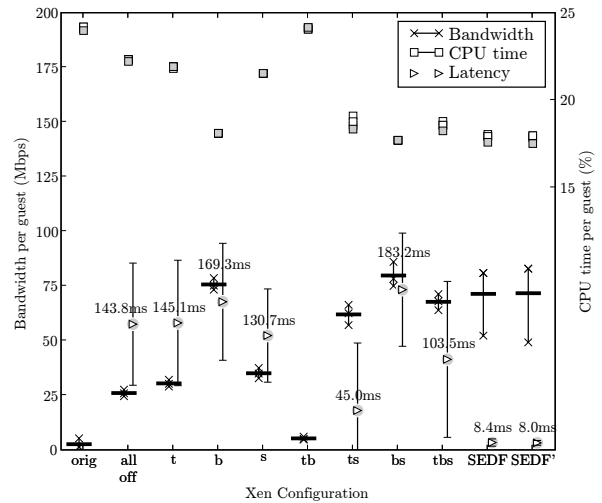


Figure 12. Bandwidth and CPU distributions with one guest both burning CPU and receiving pings (indicated in gray). The latencies for the original and tb configurations (both beyond the scale shown here) are 3.8 seconds and 593.3 milliseconds, respectively.

as this paper. They are concerned with I/O-intensive domains receiving too many processing resources, specifically, through work done on their behalf in the driver domain. In contrast, this paper shows that there are often situations in which I/O-intensive domains receive too few processing resources.

There has also been recent work on improving other aspects of virtualized I/O performance [5, 13, 15, 14, 16, 17, 20]. However, this work has largely focused on improving the efficiency of I/O operations and has not explored the impact of the scheduler on I/O performance. Scheduler improvements for I/O are likely to also benefit these innovative I/O designs.

8. Conclusions

The VMM scheduler can have a significant impact on I/O performance. Both the Credit and SEDF schedulers within Xen do a good job of fairly sharing processor resources among compute-intensive domains. However, when bandwidth-intensive and latency-sensitive domains are introduced, both schedulers provide mixed results, depending on the particular configuration.

The Credit scheduler is the default scheduler in the current version of Xen. This scheduler currently boosts the priority of idle domains when they receive events and tickles the scheduler when events are sent. It does not, however, sort domains in the run queue based on their remaining credits. This study has shown that these “optimizations” can have significant effects on I/O performance.

Two of these optimizations have generally positive effects on I/O performance. The boost optimization frequently impacts both bandwidth and latency positively, as it allows domains performing I/O operations to achieve lower response latency. Sorting the run queue based on remaining credits can have a similarly positive effect, as it allows infrequently running, but latency-sensitive, domains to run sooner.

In contrast, tickling the scheduler has mixed effects. With respect to response latency, it is neither uniformly beneficial nor harmful. However, tickling does consistently cause a wide variance in both bandwidth and latency among I/O-intensive domains. When multiple domains are performing I/O, tickling the scheduler frequently preempts the driver domain prematurely while it demul-

tiplxes packets. The scheduler effectively determines which domain to execute before it is informed of which guest domains are runnable, resulting in the observed I/O performance disparities.

Furthermore, this study has shown that latency-sensitive applications will perform best if they are placed within their own domain. The VMM scheduler can really only give a domain preferential treatment if it consistently underutilizes its processor resource allocation. However, if a latency-sensitive application shares a domain with applications that consume processor resources, then the domain is more likely to have consumed its processor resource allocation. In that case, the VMM scheduler will be unable to promptly execute the domain in response to I/O events and maintain fairness.

This study has shown that VMM schedulers do not achieve the same level of fairness for I/O-intensive workloads as they do for compute-intensive workloads. Although the existing and proposed extensions to VMM scheduling discussed in this paper may improve I/O performance to some degree, there is still much room for further improvement. Specifically, this paper has exposed two key I/O performance issues created by current VMM scheduling techniques. First, preempting the driver domain can have adverse effects on I/O performance. Second, latency-sensitive applications must be placed in their own domain to achieve the best performance. These issues both need to be addressed by future VMM schedulers.

References

- [1] E. Ackaouy. [Xen-devel] New CPU scheduler w/ SMP load balancer. <http://lists.xenproject.org/archives/html/xen-devel/2006-05/msg01315.html>.
- [2] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2006.
- [3] Advanced Micro Devices. *Secure Virtual Machine Architecture Reference Manual*, May 2005. Revision 3.01.
- [4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, Oct. 2003.
- [5] L. Cherkasova and R. Gardner. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. In *USENIX Annual Technical Conference*, Apr. 2005.
- [6] L. Cherkasova, D. Gupta, and A. Vahdat. When virtual is harder than real: Resource allocation challenges in virtual machine based IT environments. Technical Report HPL-2007-25, HP Laboratories Palo Alto, Feb. 2007.
- [7] S. Devine, E. Bugnion, and M. Rosenblum. Virtualization system including a virtual machine monitor for a computer with a segmented architecture. *US Patent #6,397,242*, Oct. 1998.
- [8] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 261–276, New York, NY, USA, 1999. ACM Press.
- [9] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the Xen virtual machine monitor. In *Proceedings of the Workshop on Operating System and Architectural Support for the On Demand IT Infrastructure (OASIS)*, Oct. 2004.
- [10] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramanian. Xen and co.: communication-aware cpu scheduling for consolidated xen-based hosting platforms. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 126–136, New York, NY, USA, 2007. ACM Press.
- [11] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in Xen. In M. van Steen and M. Henning, editors, *Middleware*, volume 4290 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 2006.
- [12] Intel. *Intel Virtualization Technology Specification for the Intel Itanium Architecture (VT-i)*, Apr. 2005. Revision 2.0.
- [13] 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*, June 2006.
- [14] A. Menon, A. L. Cox, and W. Zwaenepoel. Optimizing network virtualization in Xen. In *Proceedings of the USENIX Annual Technical Conference*, June 2006.
- [15] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the Xen virtual machine environment. In *Proceedings of the ACM/USENIX Conference on Virtual Execution Environments*, June 2005.
- [16] H. Raj and K. Schwan. Implementing a scalable self-virtualizing network interface on a multicore platform. In *Workshop on the Interaction between Operating Systems and Computer Architecture*, Oct. 2005.
- [17] H. Raj and K. Schwan. High performance and scalable i/o virtualization via self-virtualized devices. In *Proceedings of the 16th International Symposium on High Performance Distributed Computing*, June 2007.
- [18] J. Sugerman, G. Venkitachalam, and B. Lim. Virtualizing I/O devices on VMware Workstation's hosted virtual machine monitor. In *Proceedings of the USENIX Annual Technical Conference*, June 2001.
- [19] A. Whitaker, M. Shaw, and S. Gribble. Scale and performance in the Denali isolation kernel. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.
- [20] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A. L. Cox, and W. Zwaenepoel. Concurrent direct network access for virtual machine monitors. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, Feb. 2007.