IN PERFECT XEN, A PERFORMANCE STUDY OF THE EMERGING XEN SCHEDULER

A Thesis

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Ryan Hnarakis

December 2013

COMMITTEE MEMBERSHIP

TITLE:	Perfect Xen, A Performance Study of the Emerging Xen Scheduler
AUTHOR:	Ryan Hnarakis
DATE SUBMITTED:	December 2013
COMMITTEE CHAIR:	Phil Nico, PhD Associate Professor of Computer Science
COMMITTEE MEMBER:	Franz Kurfess, PhD Professor of Computer Science
COMMITTEE MEMBER:	Lynne Slivovsky, PhD Professor of Electrical Engineering

Abstract

In Perfect Xen, A Performance Study of the Emerging Xen Scheduler

by

Ryan Hnarakis

Fifty percent of Fortune 500 companies trust Xen, an open-source bare-metal hypervisor, to virtualize their websites and mission critical services in the cloud. Providing superior fault tolerance, scalability, and migration, virtualization allows these companies to run several isolated operating systems simultaneously on the same physical server [13]. These isolated operating systems, called virtual machines, require a virtual traffic guard to cooperate with one another. This guard known as the Credit2 scheduler along with the newest Xen hypervisor was recently developed to supersede the older schedulers. Since wasted CPU cycles can be costly, the Credit2 prototype must undergo significant performance validation before being released into production. Furthermore, leading commercial virtualization products, including VMWare and Microsoft Hyper-V frequently adopt Xen's proven technologies. This thesis provides quantitative performance measurements of the Credit1 and Credit2 schedulers, and provides recommendations for building hypervisor schedulers.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 What is Xen?

Xen is an enterprise-grade, open-source virtualization solution (hypervisor) that powers many websites and most cloud solutions. A hypervisor allows multiple virtual machines to run simultaneously on the same physical computer. For instance, one server could run Windows Server 2008 and Ubuntu Linux at the same time. The Xen hypervisor is capable of virtualizing x86, x86-64, and ARM instruction sets running the standard guest operating systems: Linux, Windows, and Solaris [15]. For those familiar, VMWare ESXi Server is the proprietary equivalent. Even though VMWare products are high quality and well tested, they are too costly for many applications. Xen provides a powerful, free alternative.

1.2 Who Uses Xen?

Fifty percent of the Fortune 500 companies use Xen in some capacity, whether it be for hosting websites, crunching scientific data (NASDAQ OMX), or streaming video (Netflix) [13]. Amazon Web Services (AWS), the world's largest web hosting service, and Rackspace Hosting, a close contender, rely on Xen to create isolated virtual environments for clients. The commercial

products XenServer and Oracle VM are also built atop the Xen platform. Most importantly, most people use Xen everyday without even knowing it. Yelp, TicketMaster, Shazam, PBS, Newsweek, IMDb, Sega, and Foursquare are only some of the sites running from the Xen Hypervisor. Any quantitative study of Xen stands to save industry dollars in power and equipment costs. This thesis is a step in that direction - a pause to ensure the Xen's scheduler is on the right track.

1.3 Thesis Layout

The following section expands on the Xen's background and previous work in evaluating its CPU scheduler. Then, in subsequent sections, the experimental setup and tools are described. Finally, results, further work, and conclusions follow.

Chapter 2

Background and Previous Work

2.1 Scheduling Background

Before outlining the schedulers Xen uses, we must lay the terminology groundwork and describe the goals a virtual machine scheduler seeks to accomplish.

2.1.1 Proportional Share and Fair-Share Schedulers

A CPU scheduler determines which processes, in Xen's case, virtual machines, run on the CPU at any given time. Scheduling correctly makes the scheduler virtually invisible to the user or application. Over the past couple decades, schedulers ranging in complexity were implemented with varying degrees of success. A majority of them fall in two groups: proportional share (PS) and fair-share [2].

Proportional share (PS) schedulers attempt to give CPU time to each virtual machine fairly and instantaneously. For example, if 10 CPU hungry VMs are running on one system, a PS scheduler will give each 10 percent of the CPU's time. PS schedulers are evaluated on fairness. In our example, we would see how close to 10 percent each VM maintained. If 5 of the VMs were to become inactive, the scheduler would assign the 5 running machines 20 per-

cent CPU time. Now, all 10 VMs become active again. It would make sense to award some additional CPU time to the 5 VMs that had been inactive. PS schedulers, aiming to provide instantaneous sharing among the active clients, would not award additional time. In contrast, a fair-share scheduler would give additional time to the previously inactive VMs, allowing them to catch up. Fair-share schedulers attempt to provide time-averaged, proportional sharing based on the actual use measured over long time periods [2].

2.1.2 Work Conserving and Non Work Conserving Schedulers

CPU schedulers fall into two additional categories depending on how they manage CPU idle time: work-conserving (WC) and non work-conserving. Work-conserving means that in a case of two VMs, one of these blocked, the other VM can consume the entire CPU [2]. Non work-conserving enforces caps. In other words, each VMs owns a percentage of the CPU. When the cap is set for a VM, the CPU allotment will never exceed that amount. In a case of two VMs, each VM will get up to 50 percent of CPU, but either VM will not be able to get more than 50 percent even if the rest of the CPU is idle.

2.1.3 Scheduler's 4 Goals

Virtual machine schedulers retain 4 main design goals. The first goal is fairness, which is the ability of a VM to get a fair portion of CPU resources. Fairness can be tweaked through scheduling parameters, including weight, cap, and quanta, which are described later. Fairness is more than allowing a VM to run for a given amount of time within a timeframe. Using network data transfers as an example, each bit of data sent creates more work in the future. Not allowing a network workload to run in a timely manner prevents it from receiving a fair share of CPU time when it is made available [3].

A VM scheduler's second goal is to work well with latency-sensitive workloads. Ideally, if a latency-sensitive workload uses less than its fair share of CPU, it should run as well when the system is loaded as it does when the system is idle. If a VM would use more than its fair share, then the performance should degrade gracefully [3].

The third aspect to consider is hyperthreading, an Intel feature designed to make each CPU core appear as two boosting performance. A VM running on a core by itself will have better performance than a VM sharing a core with a VM through hyperthreading. The scheduler should take this into account when determining the each VM's fair share [3].

The final scheduler goal is power efficiency. Powering down CPU cores or sockets into deeper sleep states can save power for relatively idle systems. When needed, CPU cores and sockets should be able to perform at full capacity. A scheduler needs to either implement this power-vs-performance tradeoff, or provide support for another system to do so [3]. With the scheduler terminology foundation laid, we turn to a background on Xen.

2.2 Xen History

Xen was developed at the University of Cambridge by Ian Pratt as a research project. Early versions became publicly available in 2003. XenSource Inc. supported the project for a few years until Citrix Systems acquired the company in 2007 [14].

Xen innovates extensively in the virtualization space. For example, Xen introduced the idea of paravirtualization, which is the process of informing a guest virtual machine that it is running in a virtual environment leading to significant performance gains [14]. Before paravirtualization, virtualization software simulated computer hardware to the virtual machine in a black

box approach. Virtual machines were made to believe they had unlimited access to hardware. This is a lie, as all virtual machines must share resources on the same physical machine. Paravirtualization allows for performance increases by modifying virtual machines to behave better with the underlying hypervisor. (Hypervisors are discussed in detail in the next section). This modification is usually in the form of drivers or kernel modules. A year before XenSource Inc's acquisition by Citrix, paravirtualization was adopted by major competitors Microsoft and VMWare, a sign that paravirtualization was a success. Citrix continues to release an easier to manage and set up version of Xen commercially, and some of Citrix's code trickles into the open-source Xen version [14].

Today, the Xen project is self-governed by the Xen Project Community with major contributions from large technology companies, including IBM, Intel, AMD, Hewlett- Packard, Red Hat, and Oracle. Individuals developers, may also contribute to Xen. This thesis focuses mainly on the various schedulers and their performance. It is important to note that many chefs are in the Xen kitchen. The code is constantly changing with little or no documentation. The scheduler is the product of many years of development and modification. Many conflicting accounts of the scheduler's behaviors exist [4] [2]. We reconcile conflicting accounts in the most accurate version of how the scheduler functions.

2.3 Architecture - What is a Hypervisor?

The Xen system is organized as follows. Figure 2.1 gives a high level view of Xen's architecture showing the management domain (Dom0) and three virtual machines (DomU). The management domain is essentially a virtual machine with extra permission to access the low level hypervisor. From the management domain, guest virtual machines are created, destroyed, and configured.

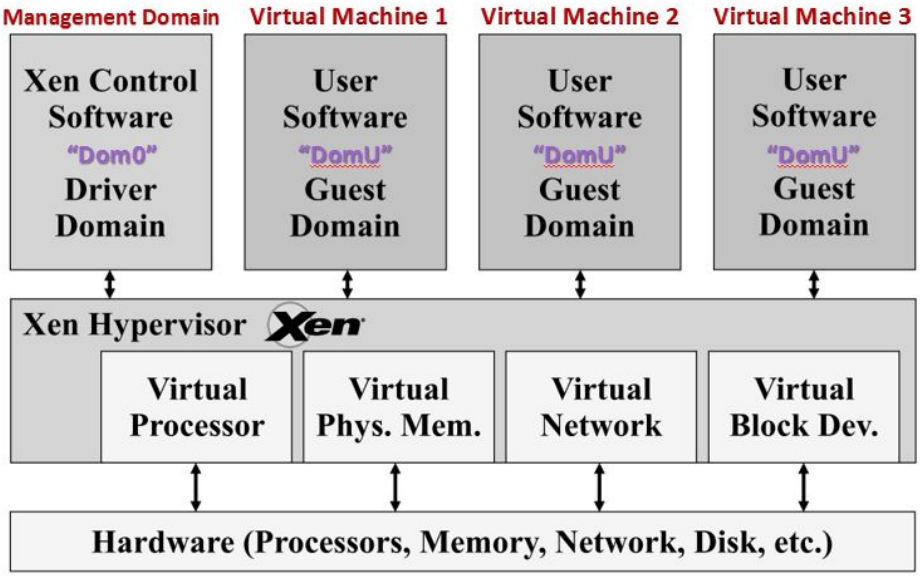

Figure 2.1: Xen Architecure

What makes Xen so fast is that no operating system stands between the virtual machines and the hardware. Only a thin software layer, called a hypervisor, mediates communication between the virtual machine and the hardware. In Xen's case, the hypervisor is a low-level, compact kernel that is most comparable to an intelligent mux [15]. The hypervisor allows real hardware resources to be connected to any number of virtual machines. *The main focus of this thesis is the scheduler contained within the hypervisor.*

2.4 Brief History and Classification of Hypervisors

Why are hypervisors necessary? Consider the scenario of Company Y developing mobile applications for Microsoft Windows Phone and the Apple iPhone. Unfortunately, Windows Phone application development is possible only on the Microsoft Windows operating system, and iPhone development is available only on the Mac OS X operating system. Company Y could choose

to invest in a dedicated Windows and Mac hardware infrastructure, or it could choose one - a hypervisor. The hypervisor makes it possible to run Windows and Mac OS on the same physical hardware as virtual machines, reducing overall equipment costs.

In the hypervisor industry, VMWare, Oracle, and Microsoft are well-known players. However, IBM created the first hypervisor called SIMMON in 1967 from research on their mainframe CP-40 system. Technology for allowing multiple users to run applications concurrently already existed before. SIMMON was the first system to allow multiple users to run applications on the same system affecting each other's work. Each user was given a virtual machine, a new term at the time. The concept was novel but suffered from a major issue; any individual virtual machine crash could bring down the entire system. Realizing this issue, IBM went back to the hypervisor design in preparation for the CP-67 system [10].

CP-67 became the first commercially available hypervisor. This time, all virtual machines were in better isolation from one another. Although each virtual machine shared the same hardware, a crash in one VM would not affect others. The technology pioneered in CP-67 became the foundation for IBM's mainframes for decades to come. IBM focused on improving robust time-sharing with hypervisors as a core mainframe technology. Schedulers were at the heart of this effort. Most of these schedulers were not very complex and scheduled in a round-robin, proportional share, work conserving fashion. Such an approach allows for a simple implementation. Even IBM's modern mainframe line, the zSeries, which powers most financial institutions, maintains backwards fully compatibility with the CP-67 tools [10].

IBM's hypervisor success led other companies to follow suit. Microsoft created a hypervisor based on its Windows operating system. VMWare and Oracle designed competing hypervisors that could run on all major operating

systems: Microsoft Windows, Mac OS X, and Linux. VMWare, IBM, Oracle, and Microsoft hypervisors fall into two categories described next.

Most modern hypervisors are considered either Type 1 or Type 2. The major difference is whether the hypervisor is closest to the hardware, as in Type 1, or one level abstracted, as in Type 2 [11]. The Xen Hypervisor, VMWare's ESX product, IBM's SIMMON/CP-40/CP-67, Microsoft Hyper-V, and Oracle VM Server are Type 1 hypervisors. When any of these systems are powered on, the hypervisor is the first piece of software to load. Once the hypervisor is booted, the virtual machines, which sit atop the hypervisor, may start. Type 1 hypervisors are commonly called bare-metal hypervisor, because they run as close to the hardware as possible and benefit from the greatest virtual machine performance. They are ideal for scientific computing and web hosting. In contrast, VMWare Workstation, VMWare Fusion, and Oracle VirtualBox are Type 2 hypervisors. When these systems are started, another operating system loads first. The hypervisor sits atop this operating system. As before, the virtual machines connect through the hypervisor. Type 2 hypervisors suffer from a performance loss due to interaction with the primary operating system one level below. Any hypervisor request would have to be approved by that OS first. Type 2 hypervisors are used typically by consumers. Both Type 1 and Type 2 hypervisors are in common use today.

Each hypervisor contains a CPU scheduler for mediating virtual machine requests for CPU time. We return our attention to the Xen Hypervisor and its CPU schedulers.

2.5 Schedulers

Figure 2.2 shows the Xen scheduler's evolution. The first and oldest scheduler is Borrowed Virtual Time (BVT), which is superseded by the Simple

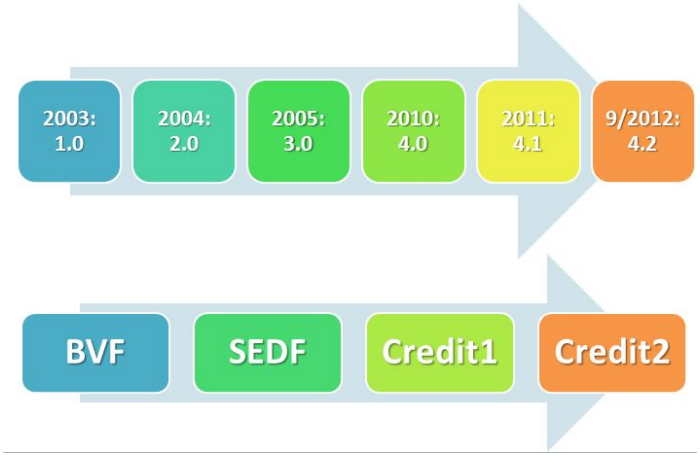

Figure 2.2: Xen Version and Scheduler History

Earliest Deadline First (SEDF), Credit1, and Credit2 schedulers. Xen allows for the selection of scheduler at boot, and the newest release offers the choice among the three most recent schedulers. Within the past two years, the Credit2 was introduced as a prototype under test. Since SEDF and Credit1 will eventually be deprecated as BVT was, Credit2 must be thoroughly validated for superior performance with Credit1. For sake of completeness and to motivate the need for Credit2, the BVT and SEDF schedulers are described next. Then, Credit1 and Credit2 are discussed in more detail.

2.5.1 Borrowed Virtual Time (BVT)

Comparison of the Three CPU Schedulers in Xen [2] fully describes the BVT and SEDF schedulers summarized below. BVT is a fair-share scheduler based on the concept of virtual time, dispatching the virtual machine with the smallest virtual time first. Virtual time is simply the amount of time a virtual machine has spent running on the CPU. The scheduler keeps track of running time in terms of a minimum charging unit (MCU), typically the frequency of clock interrupts.

The parameter C, called the context switch allowance, is the real time any currently running virtual machine is allowed to run beyond another runnable VM with equal claim on the CPU (the basic time slice or or time quantum of the algorithm). The context switch allowance is typically some multiple of the MCU. Each runnable virtual machine receives a share of CPU in proportion to its weight, the user defined priority given to the machine. To account for the weight, the virtual machine's current running time is incremented by its running time divided by weight. This makes a VM with higher weight to appear to have run for less time.

Why was SEDF not sufficient? It lacked a crucial feature important to cloud providers with multiple customer using each physical machine - non work-conserving mode (NWC). NWC allows Xen administrators to cap the CPU usage of any running virtual machine on the system. With NWC, policies, such as always giving each virtual machine 10 percent CPU no matter what, are not available. Customers want a guarantee of the percentage CPU time they will be awarded, so not having NWC is unacceptable. This limitation led to the development of SEDF.

2.5.2 Simple Earliest Deadline First (SEDF)

SEDF uses real-time algorithms to deliver guarantees. A system administrator can set each virtual machine's CPU requirements with a tuple (s, p, x). The *slice* s and the *period* p represent the CPU share that a specific virtual machine will receive. At least s units of time in each period of length p will be given. The boolean flag x indicates whether the specified virtual machine is eligible to receive extra CPU time, assuming the scheduler is in work-conserving (WC) mode. SEDF distributes this extra CPU time fairly after all runnable domains receive their CPU share. An administrator can allocate 30 percent CPU to a virtual machine by assigning the tuple as either (3 ms, 10 ms, 0) or

(30 ms, 100 ms, 0). The time granularity given in the definition of the period impacts scheduler fairness.

For each virtual machine, the SEDF scheduler tracks two additional values (d, r). d is the time which a given virtual machine's current period ends, also called the deadline. The runnable domain with the earliest deadline is picked to be scheduled next. r is the remaining CPU time in the current period.

Although SEDF accounted for the lack of work-conserving (WC) in the BVT scheduler, SEDF still has a flaw. Since SEDF implements a per CPU queue, global load balancing on multiprocessors is missing.

2.5.3 Credit1 Scheduler

Among the online wiki pages, Xen source code comments, and research papers, numerous inconsistencies exist in the operation of the Credit schedulers. One reason for this is the decentralized nature of open-source projects. Another is that the scheduling code is under active development. After extensive research and source code scrutiny, the Credit1 scheduler operation is detailed below.

From a high level, Credit1 operates on a simple currency called credit. The scheduler awards credit to each VM periodically and charges each VM to run on the CPU. Credit1 is a fair-share, priority queuing scheduler ordered in a round-robin fashion within each priority. A VM's credit and CPU usage determine to which priority it will belong. Certain conditions exist (BOOST priority) to give mostly idle VMs a chance to run without accumulating too much credit.

The authors of *Extended scheduler for efficient frequency scaling in virtualized systems* [6] further explain the algorithm. To set a VM's importance, the credit scheduler operates with two parameters: weight and cap. The weight

represents the significance of the VM as in the BVT scheduler described above, and the cap is the maximum CPU share given to a VM as an integer percentage. The scheduler is flexible can run in two modes: non work-conserving and work-conserving. When the cap is set for a VM, the CPU allotment will never exceed that amount. The scheduler is considered non work-conserving in this case. If cap is not set (null value), the corresponding VM has no CPU load limit. In this case, the scheduler is considered work-conserving, meaning that any CPU share of one VM is redistributed to others.

Each Xen computer possesses at least one CPU, and every VM requires at least one CPU, named a virtual CPU (VCPU). When the scheduler directs a VM to run, the VM's virtual CPU is connected to a physical CPU (PCPU). Although a VM typically has several VCPUs to take advantage of multiprocessor environments, we will assume each VM has one VCPU to simplify this explanation.

Xen's run queue is a simple priority queue. VMs are put in the queue based on priority: OVER, indicating the VM has consumed all its fair share of CPU resources for now, UNDER indicating credits remain, or BOOST indicating the VM transitioned from the inactive to active states (states discussed later). As a VM runs, it consumes credits. Periodically (every 10ms), a system-wide accounting thread computes how many credits each active VM has earned and grants credits based on VM weight and cap. VMs that have credit (UNDER priority) are allowed to run before any that are out of credit (OVER priority). VMs in BOOST priority run before all others. From head to tail, the run queue is ordered: BOOST, UNDER, OVER.

When inserting a VM into a physical CPU's run queue, it is put behind all other VMs of equal priority to it. Thus, movement within each priority is round robin. On each physical CPU, at every scheduling decision (after 30ms or when a VM blocks), the next VM to run is picked off of the head of the

run queue. The 30ms slice is called the time slice. When a physical CPU does not find a VM's VCPU of priority UNDER on its local run queue, it will look on other PCPUs for one. This load balancing guarantees each VM receives its fair share of system-wide CPU resources.

Each VM has one of two states, active and inactive, in addition to a priority. The state tells the scheduler to give a VM BOOST priority under certain conditions. Without the inactive state, less computationally intensive VMs, such as I/O bound VMs, would gain a huge excess of credit. To prevent this from happening, any active VM that amasses more than one time slice worth of credit (30ms) is considered inactive. When the VM becomes inactive, all of its credits are discarded and removed from the run queue.

When an inactive VM wakes, the scheduler marks it as BOOST and puts it at the tail of the run queue's BOOST priority. The VM's state remains inactive until it is caught running during one of the system-wide accounting checks, which happens every 10ms. At this point, the VM is no longer consuming a negligible amount of CPU. The scheduler will switch the state to active and assign credits. Observe that a VM in BOOST can run and block several times before being converted to active as long as the VM is not running during a system-wide accounting check.

2.5.4 Performance Comparison and Problems with Credit1

The basic performances of the BVT, SEDF, and Credit1 schedulers can be found in *Comparison of the Three CPU Schedulers in Xen* [2], although the test bed lacks multiprocessor and multicore testing. When the paper was published, the single-core, Intel Pentium III was state-of-the-art. A system with 32 cores is likely to behave differently than a system with one core. Without considering a multicore setup, the performance results may be misleading.

Around 2009, Xen researchers became aware of several problems with the Credit1 algorithm. *Scheduling I/O in Virtual Machine Monitors* [9] pointed out issues with I/O traced back to the scheduler. *Profiling and Modeling Resource Usage of Virtualized Applications* [11] discusses how the CPU (scheduler), not the NIC, frequently becomes the bottleneck when serving network traffic. Some researchers even attempted to fix the scheduler for specific types of workloads; see *Enhancement of Xen's Scheduler for MapReduce Workloads* [7]. Other issues include: not scaling well with a large number of virtual machines, not being aware of hyperthreading, maintaining a run queue sorted by priority rather than credit, using long slices that are bad for latency sensitive workloads, maintaining a run queue per core instead of L2 cache, and flopping on the boost condition. The issues were presented in detail before the Xen conference in 2009. The need for a new or patched scheduler became apparent. All these problems and possible solutions were bundled into *Xen Scheduler Status* [4].

2.5.5 Credit2 Scheduler

Credit2 is also a priority queuing scheduler, but priority is based only on each VM's credits available. The old BOOST, UNDER, and OVER priorities are not available, since run queue ordering and a new reset condition take care of latency sensitive VMs and of excessive credit accumulation.

The Credit2 scheduler addressed 3 main areas to make the scheduler more tunable, to reduce unnecessary context switching, and to fairly assign credits.

The first area added two variables, ratelimit and timeslice. Hui Lv at Intel discovered that Credit1 dispatches tens of thousands of schedules a second [12]. This means a large portion of time is wasted as the scheduler switched VMs instead of letting the work complete. The BOOST priority discussed earlier is the culprit.

Ratelimit, the first variable, combats this problem by defining the minimum number of microseconds a VM would be allowed to run uninterrupted before being context switched with the default being 1ms. Recall that the time slice for Credit1 is 30ms. Under normal operation with no VMs entering the BOOST priority, each VM would be given a whopping 30ms to run. To put this duration in context, CPU timings are usually measured in microsecond granularity. 30ms = 30,000 microseconds. Thus, 30ms is a long time. Although 30ms is perfectly acceptable for computationally intensive workloads, it is not for latency sensitive workloads.

Timeslice, the second added variable, allows for tweaking the scheduler time slice. The Xen website recommends trying lower time slices for latency sensitive workloads, but does not give any specific guidance. Nevertheless, one rule does apply; breaking the rule will lead to horrendous outcomes. The length of the timeslice (in ms) must be set higher than the length of the ratelimit. Since ratelimit and timeslice proved to be valuable, the Credit1 scheduler was modified to include them in the latest Xen releases.

The second area addressed is ordering the run queue. In Credit2, there are no BOOST, UNDER, and OVER priorities. The run queue (one per L2 cache) is ordered by each VMs' credits. When a VM wakes, the scheduler places it in the run queue by its credit available and not at the tail as in Credit1. Developers of Credit1 were concerned that ordering the run queue would produce too much overhead. The solution of Credit2 is elegant; the run queue is sorted in a lazy manner by VM insertion. By inserting VMs based on credit into the run queue, the Credit2 scheduler also ensures that mostly idle, latency sensitive VMs get a chance to run when awoken. This is Credit2's way of boosting VMs without having an explicit BOOST priority.

The third area fixed prevents any VM from accumulating too much credit by introducing a new credit reset condition. Recall that Credit1 handled this

condition by discarding the credits of a VM when it had accumulated a full time slice of credit. Credit2's reset condition affects all of the VMs' credits. Whenever the currently running VM runs out of credits, all VMs' credits are reset to the default amount. The theory behind this is presented in *Xen Development Update* [3]. In summary, the Development Update explains that the reset ensures no VM suffers from starvation from lack of credit.

As with previous schedulers, Credit2 gives the option to set a VM's weight, which is a relative level of importance to other VMs. Weight simply affects the rate at which running VMs burn credit. A higher VM weight means that a VM burns credit slower and receives more CPU time.

By addressing these 3 main areas, Credit2 addressed 3 main areas to make the scheduler more tunable, to reduce unnecessary context switching, and to fairly assign credits. We experiment to test if the improvements are noticeable.

2.6 Why Not the Completely Fair Scheduler (CFS)?

Following the description of the Xen schedulers, one might wonder why the Completely Fair Scheduler was not used instead. After all, the CFS has been accepted widely into the Linux kernel. The main reason for not using CFS is the red-black tree, which has a time complexity of O(log(N)) for insert, delete, and search. In contrast, the Credit schedulers have time complexity of O(1) To summarize CFS, the Completely Fair Scheduler places scheduled tasks in a red-black tree with used processor time as the keys. The process with the least used time works its way to left side of the tree. The scheduler picks the leftmost process in the tree to run. For a Linux system with hundreds of processes, the red-black tree is well-suited.

In Xen, each virtual machine is treated as a process from the hypervisor's perspective. Processes running within each virtual machine are not Xen's

responsibility. Because of a physical machine's RAM and CPU limitations, the Xen scheduler only has to choose from the few virtual machines running to give CPU time. A typical system configuration is limiting the number of VMs to the number of physical CPUs. In the situation of all VMs requesting CPU, the run queue would only contain 4 to 20 items. A flat run queue is a lighter weight method than a red-black tree in this scenario. CFS also lacks the cap feature of Credit1 and Credit2 that restricts virtual machines to a maximum CPU usage.

Even though the Completely Fair Scheduler's red-black tree is slightly too cumbersome for Xen, CFS does have a novel way of managing priorities, which Credit2 incorporates. CFS does not use priorities directly but instead uses them as a decay factor for the time a task is permitted to execute. Priority is equivalent to weight in Credit2. Lower-priority CFS tasks have higher factors of decay, where higher-priority tasks have lower factors of delay. Decay means that the time a task is permitted to execute dissipates more quickly for a lower-priority task than for a higher-priority task (an elegant solution to avoid maintaining run queues per priority) [5]. Recall the similar discussion about Credit2's weights above.

2.7 NAS Parallel Benchmarks

With the scheduler background aside, we turn to the question of how to best perform black-box testing. The NASA Advanced Supercomputing Division has developed a widely used set of programs designed to help evaluate the performance of parallel supercomputers. The benchmarks are derived from computational fluid dynamics (CFD) applications and consist of kernels and pseudo-applications [8]. Although the benchmarks are derived from aerospace applications, they are equally suited to evaluate virtual machine clusters. *Performance Implications of Virtualizing Multicore Cluster Machines* [10] showed

that the NAS Parallel Benchmarks are effective way to compare the performance of Xen and VMWare clusters.

The NAS Parallel Benchmarks [1] describes each of the benchmarks chosen for comparing Credit1 and Credit2 below.

1. *EP* is an embarrassingly parallel kernel, which evaluates an integral by means of pseudo-random trials. This kernel, in contrast to others in the benchmark suite, requires little interprocessor communication. EP is suited to measure raw scheduler throughput for non-latency sensitive applications and scheduler fairness.

2. *LU* is a regular-sparse, block (5 x 5) lower and upper triangular system solver benchmark that represents the computations done by a newer class of implicit CFD algorithms, typical at NASA Ames known as INS3D-LU. This benchmark exhibits somewhat less parallelism compared to the next two and more than EP.

3. *SP* solves multiple, independent systems of non diagonally dominant, scalar, pentadiagonal equations. SP and the following benchmark BT are representative of computations associated with the implicit operators of CFD codes such as ARC3D at NASA Ames. SP and BT are similar in many respects, but there is a fundamental difference with respect to the communication to computation ratio.

4. *BT* solves multiple, independent systems of non diagonally dominant, block tridiagonal equations with a (5 x 5) block size.

5. *MG* is simplified multigrid kernel benchmark. This requires highly structured long distance communication and tests both short and long distance data communication.

6. *FT* is 3-D partial differential equation solver using FFTs. FT rigorously tests long-distance communication performance.

7. *CG* is a conjugate gradient method used to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. This kernel is typical of unstructured grid computations in that it tests irregular long distance communication, employing unstructured matrix vector multiplication.

Chapter 3

Experimental Setup

Validation includes quantitative data on the operation of Credit1 and Credit2 schedulers. As the Credit2 scheduler is planned to supersede and is based on Credit1, the hope is that Credit2 would perform better in most circumstances.

3.1 Test Hardware

The main objective of black-box testing is to compare the Credit1 and Credit2 scheduler performance in an environment typical of most Xen users. Thus, a Dell rackmount server with two quad core Intel Xeon 2.50 GHz and 16 gigabytes of DDR2 RAM represents a datacenter-grade test bench. Xen in its newest alpha version does not work nicely with several consumer video cards. To properly set up Xen, video has to be directed natively from the management domain, through the hypervisor, and to the video card. Using a release version of Xen is not an option, as release versions do not contain the Credit2 scheduler. Finally, a legacy ATI card suited to handle video output was found. In stable versions of Xen, this should not be an issue.

3.2 Software Environment

From a high level, the newest Xen release (Xen v4.2.1) combined with a leading Linux distribution, Ubuntu 64-bit v12.04 Long Term Support, was chosen for hosting the experiments. The NAS Parallel Benchmarks v3.3, in concert with personally written software, orchestrated test data collection and Xen Hypervisor control.

Software versions were chosen as follows. Xen v4.2.1 contains the newest revision of the prototype Credit2 scheduler necessary for benchmarking. Ubuntu v12.04 is a long term, stable Linux distribution popular among enterprise customers and Xen users. Although the NAS Parallel Benchmarks have not changed significantly over the most recent versions, v3.3 is trusted and stable. To tie all the pieces together, a custom software suite was written and is described later.

3.3 Software Configuration

First, Ubuntu is installed natively on the server. Next, Xen is painstakingly built from source, as no prebuilt package is available for such a new Xen release. Building requires resolving several hundred dependencies manually, a process that depends equally on skill and prayer. Once Xen is configured, the original Ubuntu installation becomes Dom0, the management domain. A hypervisor now sits between Dom0 and the hardware upon rebooting.

The subsequent steps require configuring virtual machines and providing a medium for their communication.

Preliminary tests show that 3GB RAM per Ubuntu VM is sufficient to prevent swapping to physical disk. With 16GB of RAM, this allows for 4 DomU VMs with identical configuration. The Dom0 (management domain)

is given the remaining 4GB RAM to facilitate I/O operations. All VMs are updated to the newest stable Linux kernels and system packages.

To allow a medium for communication, a virtual network switch is configured linking the VMs, which allows network packets to move from one VM to another without leaving the server's physical network interface card. What is the rationale for this setup? When benchmarking the cluster, external network conditions should not be allowed to affect the results. Thus, the focus is scheduler performance. To reach the Internet if needed, the virtual switch is bridged to a physical router. Difficulties arose with trying to reach the DomU VMs. After much troubleshooting, it was found that every time a VM powers on or reboots, a randomized MAC address is assigned. This wreaks havoc on the physical router's DHCP server. A method for fixing the MAC addresses was found and all returned to perfect Zen.

Having a physical means for communication is not quite enough to run a cluster. An orderly manner for communication is needed. For the NAS Parallel Benchmarks, an open-source message passing interface known as Open MPI is recommended. An Open MPI cluster is configured with the 4 DomU VMs as slaves and the Dom0 management domain as the master.

Now that cluster communication is established, the NAS Parallel Benchmarks are ready for building and installing, which are not trivial tasks. Compilation for one server or one thousand requires equal tweaking and compiler prodding. Once the benchmarks are in place on each VM, some additional custom software is required.

3.4 Custom Software

To make Xen, the MPI cluster, and the NAS Parallel Benchmarks cooperate, a suite of custom BASH scripts were developed. The software responsi-

bilities include:

1. Switching between active schedulers
2. Verifying the active scheduler and parameters
3. Managing virtual machine power and connection states
4. Monitoring test and virtual machine readiness
5. Instrumenting test series
6. Collecting and organizing test results
7. Power cycling the entire server while maintaining test placeholders

3.5 Adjusting Benchmark Complexity

Each NAS benchmark can spawn a specified number of threads and can run in varying levels of complexity, known as problem classes. As complexity increases, processor, memory, and communication requirements increase. For a given hardware setup and benchmark, the appropriate problem class should be experimentally chosen. In increasing order of complexity, the problem classes are S, W, A, B, C, D, and E.

For the server under test, the following classes were experimentally determined to have runtimes between 100 and 1000 seconds with the Credit1 scheduler enabled. The medium length runtimes balance the needs to conserve memory, to smooth fluctuations caused by Linux processes waking, and to thoroughly stress the scheduler. Table 3.1 shows the NAS Parallel Benchmark configuration used for all experiments.

The last column in Table 3.1 shows that all benchmarks are run in 16 thread mode. Some benchmarks can only be compiled in squares (1, 4, 16,

NAS Parallel Benchmark	Problem Class	Threads
EP	Class D	16
LU	Class A	16
SP	Class A	16
BT	Class A	16
MG	Class C	16
FT	Class B	16
CG	Class A	16

Table 3.1: NAS Parallel Benchmark Problem Class Selection

25, 36... threads) and others in powers of two (1, 2, 4, 8, 16, 32... threads). Sixteen threads mode represents a reasonable, common intersection between both sets. At any given time, each of the 4 DomU VMs will want to run 4 of the 16 threads. As only 8 physical CPU cores are available the scheduler is forced to choose 8 of the 16 threads (on respective VCPUs) to run at any given time.

3.6 Testing Throughput and Fairness

For testing throughput and fairness, the Embarrassingly Parallel benchmark, or EP, is the NAS parallel benchmark best suited for the task. EP launches largely independent tasks. Under these conditions, minimal context switching should occur, and each VM should get a fair share of the CPU pie. Latency is measured in seconds for EP to complete. The shorter this time, the more throughput is possible for additional jobs. Fairness is observed qualitatively from the Xen top utility. Xen top shows the CPU utilization for each VM on the server. Since the server has 8 cores, 800 percent represents full utilization in the Xen top utility. A fair scheduler should give somewhat short of 200 percent to each of the 4 DomU VMs. The management domain Dom0 will consume a small portion of the CPU servicing I/O requests and the underlying hypervisor.

3.7 Testing Latency

Testing latency is a hypervisor scheduler is a little more difficult. For security and performance reasons, Xen does not allow any means to monitor scheduling decisions within the hypervisor layer. All monitoring must be done from the management domain VM, which is a little like attempting to move a ladder while standing on the top rung. Instead of taking the average time the scheduler takes to respond to individual VMs, the total runtime for latency sensitive tests is recorded. The NAS Parallel Benchmarks LU, SP, BT, MG, FT, and CG are structured in a way that force a longer total runtime if individual latencies are high.

Chapter 4

Results

4.1 Overview

The Credit1 and Credit2 face-off is done, and the outcomes of tests to measure scheduler latency, throughput, and qualitative fairness are described in this chapter. Credit2 is still under active development, so these test results may not accurately represent the scheduler in v4.2.1.

Note Figure 4.1 showing the results of the NAS Parallel Benchmarks. On the horizontal axis, each benchmark is shown as tested with the Credit1 and Credit2 schedulers. The vertical axis indicates the corresponding averaged runtimes in seconds over several identical trials. A shorter bar indicates a better result.

Averages hide an important piece of information: how consistent the data is. Figure 4.2 shows the runtimes' sample deviations, which are calculated as the standard deviation divided by the average and displayed on the vertical axis. Similar to Figure 4.1, the horizontal axis shows each benchmark run in Credit1 and Credit2 modes. With schedulers, consistency is desirable, so a lower sample deviation is better in most cases.

For all the latency sensitive benchmarks, that is all besides EP, Credit2 is the clear winner in runtimes and sample deviations. In other words, run-

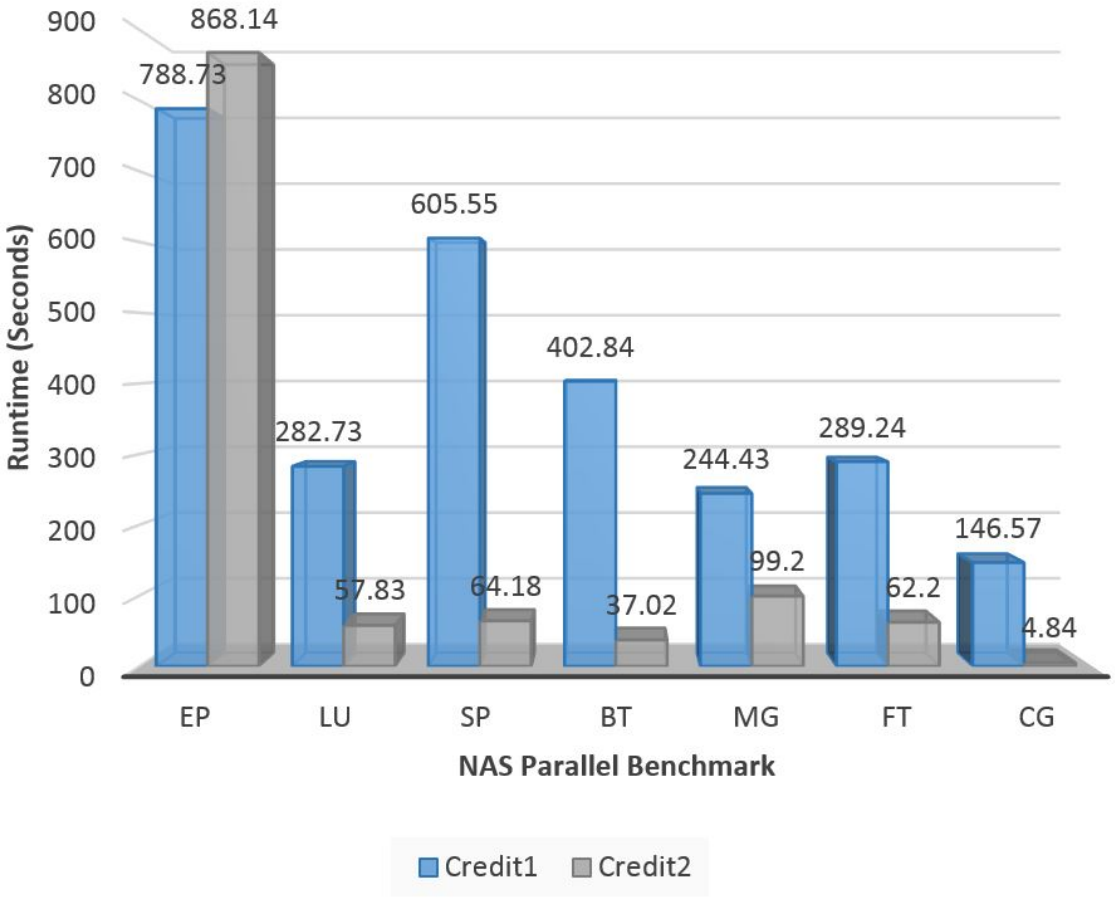

Figure 4.1: Average Runtimes for Each Benchmark

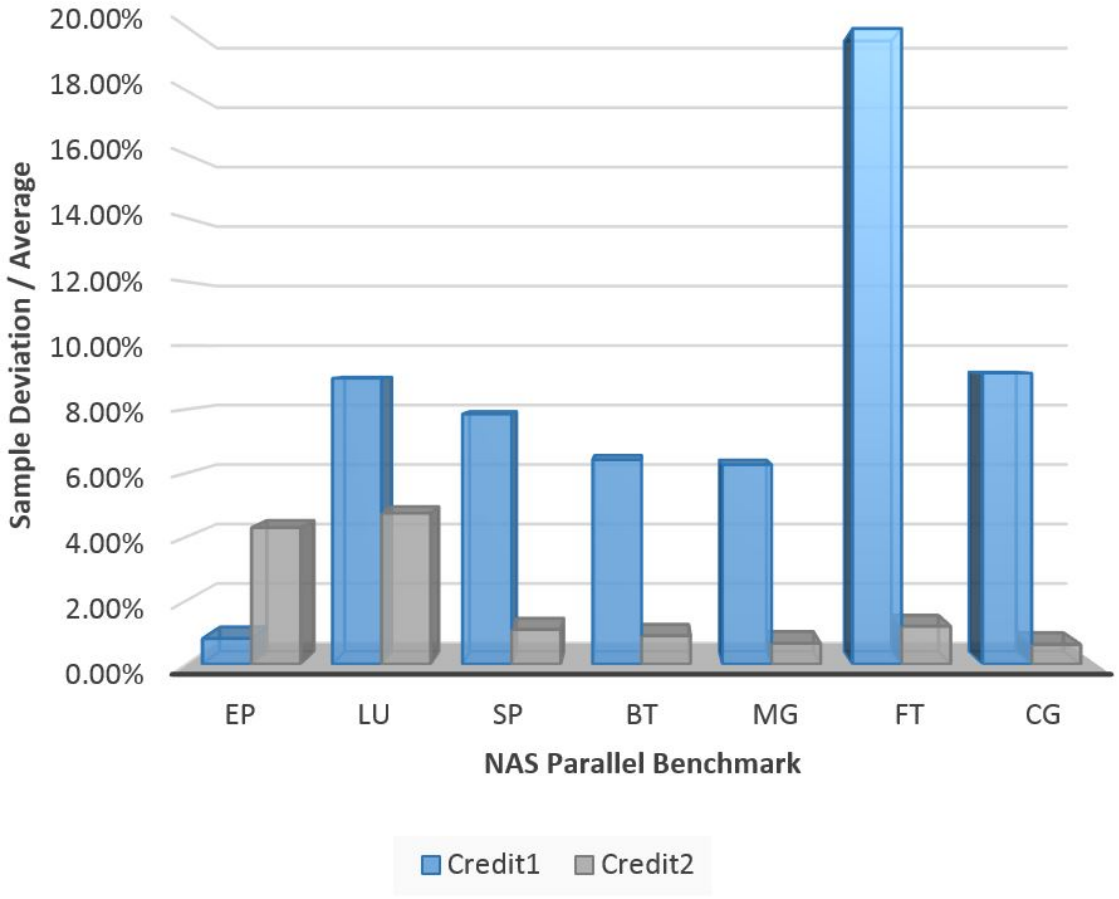

Figure 4.2: Sample Deviations

times and sample deviations are lower for all these benchmarks. Not only did the latency sensitive benchmarks finish sooner on Credit2, their performance was more consistent on subsequent runs than on Credit1. Latency sensitive applications include photo editing, music encoding, network-heavy programs. This consistency further translates to more fair scheduling decisions and is observable through the Xen top utility. Credit2 excels for several reasons in this area. First, Credit2 is aware of L2 shared caches, which saves time retrieving data from memory. Second, Credit2's hyperthreading awareness gives it an extra boost. Finally, the scheduling quanta ratelimit prevents excessive context switching.

As mentioned, the EP benchmark, more characteristic of sustained CPU bound workloads, performed slightly better on Credit1 most of the time (about 9 percent better). It should be noted that the EP benchmark is more parallel than most applications, even scientific, will ever achieve. Little interprocessor communication is required for EP, which makes it a fairly contrived case. When designing a scheduler, it is common to test the case in which all processes or VMs request resources all the time. Credit1 was built around this notion, as evident for Credit1's excellent EP and dismal latency-sensitive results. Credit2 may perform better on EP if various scheduler quanta, such as ratelimit and timeslice are optimized.

4.2 Limitations

Experimental setups, like the one used to generate these results, inherently have limitations. Most of the limitations here arise from the scope of testing. For example, no real workloads, such as web hosting loads, were tested. Websites can tend to have bursts of traffic, which were not tested. Neither did we look at an important criterion affecting critical websites: reliability. We assumed Credit1 and Credit2 are sufficiently reliable to provide the performance

seen in our initial results.

As discussed earlier, Hui Lv at Intel discovered Credit1 engaged in excessive context switching. Our experiment does not directly show the rate at which context switching occurs. Currently, Xen does not provide instrumentation for this value. It would have been interesting to verify that Credit2 had a lower context switch rate than Credit1. Credit2 also claimed hyperthreading awareness. The results above were run with the default CPU settings. Disabling hyperthreading may have led to different benchmark durations.

The NAS Parallel Benchmarks were run on a single system with 8 CPU cores. Traditionally, the benchmarks have been run on large clusters with frequent network communication. The tests run in this experiment ignored network dynamics typcial of large clusters of machines, which frequently have far in excess of 8 CPU cores per machine. Intel Xeon server CPUs have up 15 cores (30 threads hyperthreaded). With up to 4 CPUs in one system, the scheduler would have up to 120 threads to contend with. Credit2 was supposed to excel on systems with a high number of cores available.

A final limitation of our results revolves around the operating system choice. All of the virtual machines tested were Linux based atop Xen, a Linux hypervisor. Could Windows virtual machines with different internal schedulers affected the Xen scheduler differently? Our experiment does not cover this possibility, which could be a place to start for future work.

Chapter 5

Conclusion

With the results tallied, the new Xen Credit2 scheduler stands as a much needed replacement for the existing Credit1 scheduler. Credit2 proves an algorithm well suited for running virtual machines. The new scheduler runs faster in most situations and behaves more predictably. Nevertheless, a certain highly parallel load with few interrupts leads to slightly worse performance on this new algorithm. The Xen open-source community should investigate and provide a fix if beneficial to overall performance.

After dissecting Xen's Credit1 scheduler, a pitfall became clear: allowing head-of-the-line privileges. When designing a general purpose scheduler, an algorithm that permits head-of-the-line privileges for any process is risky and should be avoided if possible. Credit1, plagued by BOOST priority, can clog the entire run queue with BOOST VMs under some conditions creating context switching overhead. The BOOST head-of-the-line privilege made Credit1 unpredictable and inefficient.

These findings impact more than hundreds of enterprises using the Xen hypervisor. As an upstream technology, Xen affects all other commercial virtualization solutions, including VMWare and Microsoft's products. With the Internet moving increasingly toward virtualization, hypervisor advancements are critical to the Internet's performance and stability. Xen's Credit2 scheduler

is yet another step in the right direction for this maturing technology.

Chapter 6

Future Work

6.1 Timeslice and Ratelimit

The tested Xen Hypervisor version did not allow for modification of the Credit2 scheduler parameters timeslice and ratelimit as an end user. Tweaking these parameters for different workloads would help Xen choose better default values and recommend specific values. Modification of the parameters in source code requires a full recompilation of the hypervisor and kernel and has unintended consequences. As soon as possible, Xen should allow parameters to be changed in Credit2 as it does in Credit1. Then, a complete evaluation of the paramenters can be made.

6.2 Tweak Credit2 for better EP Performance

Once the Credit2 parameters are accessible, an analysis as to why EP performed worse on Credit2 should be explored. EP may not be typical of most applications. In that case, the Credit2 scheduler may not require modification. EP could be a parasitic case, which may require more tuning of the Credit2 scheduler.

6.3 Study the IO Scheduler

In many applications, such as MySQL server, performance is bound by disk or flash memory IO. It would be worth exploring Xen's current IO scheduler and its interaction with the Credit CPU scheduler. Additionally, a study of system overprovisioning should be performed. Overprovisioning would entail running enough VMs to run low on RAM. Once VMs started swapping RAM to physical disk, the IO schedulers actions might affect the CPU scheduler's performance.

Bibliography

[1] D. H. Bailey. The nas parallel benchmarks. In *Intl. Journal of Supercomputer Applications*, pages 66–73. SAGE Publications, 1991.

[2] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51, Sept. 2007.

[3] G. Dunlap. Xen development update, 2009.

[4] G. Dunlap. Xen scheduler status, 2009.

[5] Inside the linux 2.6 completely fair scheduler. http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/, June 2013.

[6] C. M. Kamga, G. S. Tran, and L. Broto. Extended scheduler for efficient frequency scaling in virtualized systems. *SIGOPS Oper. Syst. Rev.*, 46(2):28–35, July 2012.

[7] H. Kang, Y. Chen, J. L. Wong, R. Sion, and J. Wu. Enhancement of xen's scheduler for mapreduce workloads. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 251–262, New York, NY, USA, 2011. ACM.

[8] Nas parallel benchmarks. http://www.nas.nasa.gov/publications/npb.html, June 2013.

[9] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling i/o in virtual machine monitors. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '08, pages 1–10, New York, NY, USA, 2008. ACM.

[10] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan. Performance implications of virtualizing multicore cluster machines. In *Proceedings of the 2nd workshop on System-level virtualization for high performance computing*, HPCVirt '08, pages 1–8, New York, NY, USA, 2008. ACM.

[11] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Profiling and modeling resource usage of virtualized applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 366–387, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[12] Credit Scheduler. http://wiki.xen.org/wiki/, June 2013.

[13] Citrix XenServer Certified on IBM System x and BladeCenter Servers. http://www.citrix.com/news/announcements/mar-2011/citrix-xenserver-certified-on-ibm-system-x-and-bladecenter-servers.html, December 2013.

[14] Xen History. http://www.xen.org/community/xenhistory.html, June 2013.

[15] Why the Xen Project. http://www.xenproject.org/users/why-the-xen-project.html, December 2013.