Technical Report CS10-01

An Empirical Evaluation of Methods for Improving Efficiency in Xen Virtual Machine CPU Scheduling

James Devine

Submitted to the Faculty of
The Department of Computer Science

Project Director: Dr. Robert D. Cupper
Second Reader: Dr. Gregory M. Kapfhammer

Allegheny College
2010

I hereby recognize and pledge to fulfill my responsibilities as defined in the Honor Code, and to maintain the integrity of both myself and the college community as a whole.

James Devine

JAMES DEVINE. An Empirical Evaluation of Methods for Improving Efficiency in Xen Virtual Machine CPU Scheduling. (Under the direction of Dr. Robert D. Cupper.)

ABSTRACT

Virtualization is a technique for running multiple operating systems on a single host machine. This is done though a lightweight operating system called a hypervisor that allocates resources to each guest operating system. Xen is one such hypervisor that is both open source and free. The Xen hypervisor offers two different virtual machine scheduling algorithms, the credit scheduler (which is the default) and the Simple Earliest Deadline First (sEDF) scheduler. Virtual machine performance is sensitive to the hypervisor's scheduling of CPU resources. This research examines various configurations of the credit and sEDF schedulers for specific work loads by constructing three case study applications for experimentally determining and evaluating the performance of each of the scheduling algorithm configurations for each of the three workloads. The results of the study suggest that the default CPU scheduling algorithm in the default configuration does not provide any performance guarantees. Further, various configurations of the sEDF CPU scheduler consistently outperformed the credit scheduler for some of the case study applications.

Acknowledgments

First and foremost I would like to thank my first reader, Professor Cupper. He has severed as colleague, friend, and mentor throughout the entire process of this work. He was always there to provided constructive criticism and ultimately has a large share in the success of this work.

I would also like the thank The MITRE Corporation for providing a challenging and rewarding internship experience over the last two years. Through the course of the internship I was able to firmly plant my interest in virtualization and gain invaluable hands on experience that was crucial to the success of this work. The list of specific individuals at MITRE who have help guide me include: Ed Shrum, Dave Polete, Paul Barr, and David Kaplan.

Contents

Acknowledgments **iv**

List of Tables **x**

List of Figures **xiii**

1 Overview **1**

1.1 Virtualization 1

1.1.1 Definition 1

1.1.2 Motivation 2

1.1.3 History 4

1.1.4 The Xen Open Source Hypervisor 5

1.2 CPU Scheduling 6

1.2.1 Measuring Performance 7

1.2.2 Xen CPU Scheduling Algorithms 8

1.2.2.1 The sEDF Scheduler 9

1.2.2.2 The Credit Scheduler 9

1.2.3 Scheduling Concerns 10

1.3 Thesis Statement 11

1.4 Contributions of Work 11

1.5 Looking Ahead 12

2 Related Work **14**

2.1 Xen and the Art of Virtualization 14

2.2 Xen CPU Scheduling Research 15

3 Virtual Machine Architecture **17**

3.1 Hardware 17

3.2 Workloads 17

3.2.1 Web Application 18

3.2.2 VPS Application 19

3.2.3 Windows Domain 20

3.2.3.1 Active Directory 21

3.2.3.2 Exchange 21

3.2.3.3 SharePoint 22

3.2.3.4 MSSQL 22

3.3 Test Machine 22

3.4 Network Overview 22

4 Experiment Design **25**

4.1 Web Application Workload Experiment 25

4.1.1 Application Design 26

4.1.2 Running the Experiment 27

4.2 VPS Workload Experiment 27

4.2.1 Running the Experiment 28

4.3 Windows Domain Workload Experiment 28

4.3.1 Running the Experiments 29

4.4 Results Analysis 30

4.4.1 JMeter 31

4.4.2 LoadGen 32
4.4.3 Xenmon 33

5 The Results: Analysis and Conclusions 34

5.1 Introduction 34
5.2 Web Application Results 36
5.3 VPS Results 46
5.4 Windows Domain Results 46
5.4.1 Exchange Results 46
5.4.2 MOSS Results 46
5.5 A Review of Claims and Contributions 56
5.6 Threats to Validity 57
5.7 Conclusion 57

6 Future Work 59

A Xen Commands 62

A.1 Xen Installation 62
A.2 Creating a VM 63
A.3 Cloning a VM 63
A.4 Increase the Number of Loopback Devices 63
A.5 Set the Xen CPU Scheduler Type 64
A.6 Determining the Scheduler Configuration 64
A.7 Xentop 64
A.8 Viewing the Running Virtual Machines 65

B Tools 67

B.1 JMeter 67

B.2 LoadGen 67

B.3 Pageant 68

B.4 Plink 68

B.5 Ubuntu 68

B.6 virt-manager 69

B.7 Xenmon 69

B.8 Xen 69

C Source Code **70**

C.1 Test Configuration Files 70

C.1.1 Web Application JMeter Configuration File 70

C.2 Test Scripts 74

C.2.1 Web Application sEDF Tests Powershell Script 74

C.2.2 Web Application Credit Tests Powershell Script 77

C.3 Web Application Source Code 80

C.3.1 Main Website 80

C.3.2 SQL Setup Script 85

C.3.3 Powershell Script to Execute the SQL Setup 88

C.3.4 SQL Teardown Script 89

C.3.5 Powershell Script to Execute the SQL Teardown 90

C.3.6 SQL Configuration File 90

C.4 JMeter Log File Reduction 91

C.4.1 JMeter Log File to CSV Converter 91

C.4.2 Script to Run the JMeter Converter 94

C.5 R Code 95

C.5.1 Xenmon Output Processing 95

Bibliography **97**

List of Tables

3.1 Virtual System Specifications. An overview of the testbed system specifications. 18

3.2 Test Computer Specifications. An overview of the test computer specifications. 23

4.1 Web Application sEDF Scheduler Configurations. An overview of the slice and period used for the sEDF scheduler. The value in parenthesis indicates the name used to refer to that scheduler configuration. . . . 26

4.2 Windows Domain sEDF Scheduler Configurations. An overview of the slice and period used for the sEDF scheduler. 30

4.3 JMeter Log Entry Parameter. A description of each of the JMeter parameters captured. 32

5.1 Web Application Throughput Results. An overview of the web application throughput results. 39

5.2 Web Application Throughput Standard Deviation. An overview of the standard deviation of the throughput values obtained in the web application tests. 39

5.3 Web Application Throughput 95% Confidence Intervals. An overview of the 95% confidence interval of the results obtained in the web application tests. Each value indicated in the table is a $\pm$ value from the mean. 42

5.4 Web Application Response Time Results. An overview of the web application response time results. 42
5.5 Web Application Response Time Standard Deviation. An overview of the standard deviation of the response times obtained in the web application tests. 43
5.6 Web Application Response Time 95% Confidence Intervals. An overview of the 95% confidence interval of the response times obtained in the web application tests. Each value indicated in the table is a ± value from the mean. 43
5.7 Web Application Apache VM CPU Time. An summary of the percentage of total runtime that the Apache VM spent in the blocked, running, and waiting states. 44
5.8 Web Application NFS VM CPU Time. An summary of the percentage of total runtime that the NFS VM spent in the blocked, running, and waiting states. 44
5.9 Web Application MySQL VM CPU Time. An summary of the percentage of total runtime that the MySQL VM spent in the blocked, running, and waiting states. 45
5.10 Web Application Domain0 CPU Time. An summary of the percentage of total runtime that Domain0 spent in the blocked, running, and waiting states. 45
5.11 MOSS Throughput Results. An overview of the MOSS throughput results. 49
5.12 MOSS Throughput Standard Deviation. An overview of the standard deviation of the throughput values obtained in the MOSS tests. . . . 52

5.13 MOSS Throughput 95% Confidence Intervals. An overview of the 95% confidence interval of the results obtained in the MOSS tests. Each value indicated in the table is a ± value from the mean. 52
5.14 MOSS Response Time Results. An overview of the MOSS response time results. 53
5.15 MOSS Response Time Standard Deviation. An overview of the standard deviation of the response times obtained in the MOSS tests. . . 53
5.16 MOSS Response Time 95% Confidence Intervals. An overview of the 95% confidence interval of the response times obtained in the MOSS tests. Each value indicated in the table is a ± value from the mean. . 54
5.17 MOSS VM CPU Time. An summary of the percentage of total runtime that the MOSS VM spent in the blocked, running, and waiting states. 54
5.18 MOSS MSSQL VM CPU Time. An summary of the percentage of total runtime that the MSSQL VM spent in the blocked, running, and waiting states. 55
5.19 MOSS Domain0 CPU Time. An summary of the percentage of total runtime that Domain0 spent in the blocked, running, and waiting states. 55

List of Figures

1.1 An Abstract View of The Xen Architecture. The Xen Hypervisor is the underlying software that directly accesses the CPU and memory. Domain0 is a special virtual machine that has direct access to the physical I/O resources of the host. Two guest virtual machines are shown to the right of Domain0, one paravirtualized (PV) and one fully virtualized (HVM) [10]. 6

1.2 The States of a Process. A flowchart of the three possible states of a process: running, ready, and blocked. 7

3.1 Web Application Workload Architecture. A high-level view of the web application workload. 19

3.2 VPS Hosting Workload Architecture. A high-level view of the VPS application workload. 20

3.3 Windows Domain Workload Architecture. A high-level view of the windows domain application workload. 21

3.4 Network Architecture. An overview of the network architecture created for experimentation. 24

4.1 Pseudo Code for the Web Application. A basic outline of the web application design. 27

4.2 Pseudo Code for the Web Application Workload Testing Script. A basic outline of the script used to run the web application workload experiment. 27

4.3 Pseudo Code for the VPS Workload Testing Script. A basic outline of the script used to run the VPS workload experiment. 28

4.4 Pseudo Code for the Windows Workload Testing Script. A basic outline of the Windows PowerShell scripts that were used for the Window domain workload experiments. 30

4.5 JMeter Log Entry. A sample of a JMeter log entry. An explanation of each field is provided in Table 4.3. 31

5.1 Web Application Throughput Graph. A graphical depiction of the average throughput for each scheduler configuration under each user load with 95% confidence intervals. 40

5.2 Web Application Response Time Graph. A graphical depiction of the average response time for each scheduler configuration under each user load with 95% confidence intervals. 41

5.3 MOSS Throughput Graph. A graphical depiction of the average throughput for each scheduler configuration under each user load with 95% confidence intervals. 50

5.4 MOSS Response Time Graph. A graphical depiction of the average response time for each scheduler configuration under each user load with 95% confidence intervals. 51

A.1 `sched-credit` Command. An example of the `xm sched-credit` command output. 65

A.2 `xentop` Command. An example of the `xentop` command output. . 65

A.3 `xm list` Command. An example of the `xm list` command output. 66

Chapter 1

Overview

"...performance almost always matters. And I absolutely detest the fact that people so often dismiss performance concerns so readily"
—Linus Torvalds

This chapter describes virtualization and gives a historical background to provide perspective. After establishing the background, a motivation for the use of virtualization is presented followed by the thesis statement.

1.1 Virtualization

This section gives a basic overview of virtualization and its history. It also provides a motivation for the use of virtualization and introduces a particular virtualization system, the Xen hypervisor.

1.1.1 Definition

Virtualization is a term that means many things to many people. In general it refers to the process of abstracting a resource. In terms of operating systems, *virtualization* is referred to as software technique that allows for multiple operating systems to be run concurrently on a single computer. Hardware resources are abstracted by a lightweight operating system referred to as a hypervisor. The term hypervisor is often used interchangeably with the term virtual machine monitor (VMM). The lightweight operating system has direct access to the physical resources and allocates them to some number of virtual operating systems, called virtual machines (VMs), running "on top" of the hypervisor. Through the use of operating system virtualization, many operating systems of the same or differing types, e.g. Linux and Windows, can be run concurrently using the same physical hardware.

The concept of virtualization arose in the 1970's with the advent of the IBM 370 operating system. Within the last five years virtualization has become increasingly popular due to increasing hardware speed coupled with decreasing prices. This has led to the observation that, on average, computers are operating at just 10% to 20% hardware utilization.

1.1.2 Motivation

Modern central processing units (CPUs) have become increasingly powerful in recent years. Moore's Law, which states that the number of transistors on a chip will double every two years, has continued to hold since Moore first stated it in 1965 [4]. Given this trend, CPU manufactures have moved away from the strategy of increasing the clock speed of processors to adding more and more processors (referred to as cores) to single chips. This trend has extended to even the low-end workstation processor line to the point that today it is hard to find CPUs being manufactured with less than two cores. In many cases software applications do not take full advantage of the processing power of a single core, let alone multiple cores, which can lead to a large degree of unused CPU resources.

The trend of having unused resources is a concern to businesses running multiple servers. If eight servers are running at 10% average utilization, then they could, theoretically, be consolidated to a single server with an 80% utilization plus the overhead of the hypervisor. In the case of the Xen hypervisor, this overhead is approximately 3% [11]. The motivation for server consolidation has caused a rapid increase in the use of virtualization by businesses and other institutions that have the need to maintain many servers. In addition to a reduced hardware acquisition cost, there are cost savings in the form of reduced energy, cooling, infrastructure, and maintenance. Fewer servers means less servers to power and cool. The use of virtual networking, providing network connectivity via software instead of hardware, allows for further cost savings by reducing the amount of physical networking hardware.

There are several other benefits to virtualization including easier reallocation of resources, faster provisioning of new systems, and higher availability. Virtual machines can be provisioned much more quickly than a typical server since the hardware is already in place, which just leaves the task of installing an operating system. The installation of an operating system itself can be eliminated though the use of template virtual machines from which a fully function operating system can be deployed quickly. In addition, when a virtual machine needs additional resources, a hardware upgrade is not required. A virtual machine can simply be allocated more CPU and memory resources though adjustment of a configuration file. Further, flexibility is augmented though mechanisms like live migration which allows a running virtual machine to be migrated between virtual servers in a matter of seconds without the need to power down. Live migration can be automated to allow for load balancing across virtual servers in a cluster. It can also facilitate high availability by detecting a failing host server and migrating VMs to a healthy server.

However, there is a price for all these benefits that virtualization offers. Virtual machines do not have direct access to hardware resources like a standalone system does. The hypervisor must therefore contend with the complex task of scheduling access to each physical resource (disk, network, memory, and CPU) due to the inability of many devices to support concurrent operations. For example, a CPU can only execute one instruction per core in any given clock cycle. This fact is further complicated because many of the resources are scheduled by each individual VM's operating system. In context of the CPU, a virtual machine cannot run any processes until it has access to a CPU. A virtual machine will not have access to the CPU until the hypervisor's CPU scheduler has moved the virtual machine from the wait queue to a running state. Once a virtual machine is running, it can run processes that are in the wait queue of the virtual machine's CPU scheduler.

1.1.3 History

Operating system virtualization has existed for quite some time. IBM's VM/370 provided the capacity to concurrently run multiple operating systems in 1972 [6]. Virtualization has remained a topic of research since that time. The main restraint was that the hardware of the time did not have sufficient excess capacity to support the full virtualization of operating systems. Processors were slow and the only way to run truly parallel operations was to use multiple CPUs in a machine. Memory was also prohibitive expensive. Aside from costs, hypervisor technology was in its infancy. This remained the case for many years until operating system virtualization became more technologically and financially viable.

VMware was one of the first companies to start offering a hypervisor for commodity hardware in 1999 with their release of VMware Workstation [14]. The Xen open source hypervisor project followed several years later, beginning in 2003 [1]. The recent increasing use of virtualization has prompted companies like Citrix and Microsoft to release their own hypervisors, XenServer (based on Xen) and HyperV, respectively, to compete with VMware.

1.1.4 The Xen Open Source Hypervisor

Xen is an open source hypervisor that was first released in 2003 [1]. Initially, the hypervisor only supported paravirtualization which restricted Xen to supporting only virtual machines with a modified version of the host Linux kernel. This limited Xen to running a limited set of operating systems. Eventually, support for full virtualization was included which allowed Xen to support running virtual machines with non-Linux operating systems, such as those offered by Microsoft and Sun [10].

The Xen architecture, depicted in Figure 1.1, is different from that of Microsoft and VMware in that I/O is not handled by the hypervisor. The Xen hypervisor is primarily responsible for allocating the CPU and memory and has direct access to the physical hardware. All disk and network I/O is handled though a special virtual machine referred to as Domain0 (virtual machines are referred to as *domains* in Xen). Domain0 is the only virtual machine that has direct access to physical I/O hardware resources. All of the other virtual machines access disk, network, universal serial bus (USB) devices, and all other peripherals though Domain0.

The Xen hypervisor was chosen for this work due to its open source nature. Xen is, in fact, the only x86 open source hypervisor. The Xen hypervisor is also quite unique in that it has the ability to change the CPU scheduling algorithm with the addition of a boot parameter. This functionality is discussed in the next section.

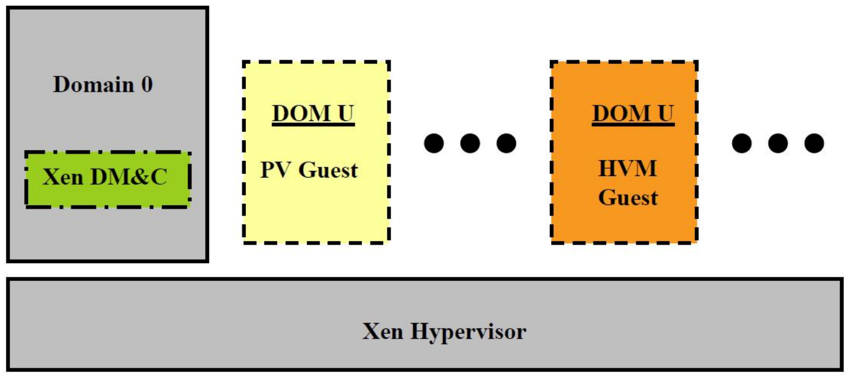


Figure 1.1: An Abstract View of The Xen Architecture. The Xen Hypervisor is the underlying software that directly accesses the CPU and memory. Domain0 is a special virtual machine that has direct access to the physical I/O resources of the host. Two guest virtual machines are shown to the right of Domain0, one paravirtualized (PV) and one fully virtualized (HVM) [10].

1.2 CPU Scheduling

One of the largest concerns with virtualization is resource allocation. While there may be the appearance of ample idle resources in a physical machine, improper scheduling of access to resources can cause problems. Modern operating systems utilize time-sharing to allow for processes to run concurrently (or appear to run concurrently in the case of a machine with only one CPU). This time sharing is largely facilitated by a CPU scheduler that must deal with the fact that at any given time only one process per CPU core can be running. CPU scheduling is a very complicated process and is paramount to the efficient functioning of an operating system.

Regardless of the CPU scheduling algorithms that an operating system uses, there are three basic states that a *process* (a term used to describe a program in execution) many be in: running, ready, or blocked [12]. A process that is **running** has access to CPU resources. A process that is **ready** is waiting for access to CPU resources, but cannot access a CPU because another process is running. A process that is **blocked**

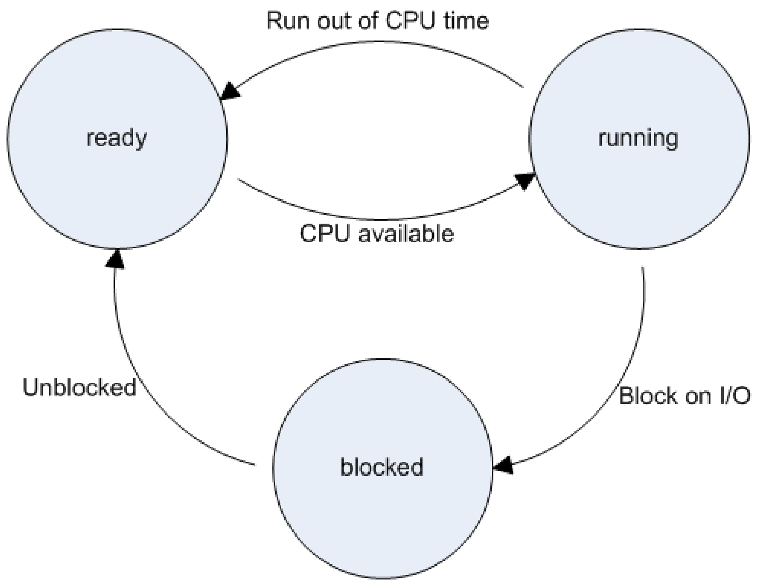


Figure 1.2: The States of a Process. A flowchart of the three possible states of a process: running, ready, and blocked.

is waiting for some external event, such as I/O, to occur. The states that a process can be in are depicted in Figure 1.2. These three states account for the total time that a process is in execution.

1.2.1 Measuring Performance

The performance of a CPU scheduling algorithm can be assessed in terms of throughput and/or response time. An efficient CPU scheduling algorithm has a low a response time with a high level of throughput.

Throughput is generally expressed as the amount of work performed over a given unit of time, i.e., $work/time$. A CPU scheduling algorithm that performs well will accomplish a large amount of work in a given amount of time.

Response time in CPU scheduling is the time between when a command is issued and a result is obtained [12]. For example, in interactive systems, such as a personal computer, the time between when a user types a letter on his keyboard and the letter appears on their screen would be the response time. The measure of response time when a CPU scheduler is scheduling the CPU for VMs is slightly different. There is no direct way to measure response time in such a fashion at the hypervisor level due to the fact that a virtual machine process encapsulates all of the processes running on a particular VM.

A better methodology for calculating response time for VM processes at the hypervisor level is to look at the amount of time a process is in the ready state waiting for access to a CPU. Using this metric provides an understanding of how much time a VM spends waiting for access to a CPU. Much like a traditional measure of response time, lower is better. Therefore an efficient CPU scheduling algorithm will reduce the amount of time a process is in the ready state. The remainder of this paper will use this metric whenever response time is referred to.

1.2.2 Xen CPU Scheduling Algorithms

The Xen hypervisor supports two CPU scheduling algorithms, the simple earliest deadline first (sEDF) scheduler and the credit scheduler. The credit scheduler is the default scheduler that is enabled with the default installation of Xen. The sEDF scheduling algorithm can be used by setting the `sched` kernel boot parameter (see Appendix A.5 for more details). Each CPU scheduling algorithm schedules CPU time for the virtual CPUs (VCPU) for each VM.

1.2.2.1 The sEDF Scheduler

The sEDF (Simple Earliest Deadline First) scheduler delivers guaranteed CPU resources to a VM. For each VM, a slice s and period p are set in milliseconds using the `xm sched-sedf` command (for more details see Appendix A.5). Each VM is guaranteed to receive s CPU time over each p. There is an optional boolean flag that specifies whether a VM can receive extra CPU time, that is time not guaranteed to other VMs during a specific p.

The sEDF algorithm keeps track of two additional values, the time a VM's current period ended, d, and remaining CPU time, r. The value d is used to determine what VM to schedule next and the value r keeps track of the remaining time that a VM has in a given period [2]. The VM with the earliest deadline i.e., the VM that has gone the longest without access to the CPU, and still has remaining time (a positive r value) will be scheduled first.

1.2.2.2 The Credit Scheduler

The credit scheduler is a "proportional fair-share CPU scheduler" [16]. The scheduler tracks the "credit" of each VCPU that is scheduled for CPU time. A VCPU can either be over credit (exceeding its fair share of the CPU) or under credit (not exceeding its fair share of the CPU). Every time a VCPU has access to the CPU it uses credits, if it is already over credits its credit value becomes negative. Priority is given to the VCPUs that are under credit. At a predetermined time the credit of each VCPU is reset.

The credit scheduler allows for a weight and cap value to be placed on each VM. The weight value has a default setting of 256, with greater weight values specifying a greater scheduling priority. For example, a VM with a weight of 512 will have twice the scheduling priority as a VM with a weight of 256. The maximum value for weight is 65535 [16]. The cap value determines the maximum number of CPU cores that a VM can use and by default is set to 0, which indicates no limit. Each CPU core is expressed as 100%, so a VM with a cap of 200% can use a maximum of two CPUs.

1.2.3 Scheduling Concerns

The CPU scheduler has much to do with the performance of the various VMs running on a virtual server. The scheduler is controlling VCPU access to CPUs, which are not the actual processes that are running on the VMs, but rather an encapsulation of the VM's processor. When a VCPU has access to a CPU, the underlying OS running on the VM can use its own scheduler to schedule processes on the VM. This construct adds a delay that is not present on physical hardware. If the hypervisor CPU scheduler does not operate efficiently, it can affect all of the VMs running on the server.

A big concern with CPU schedulers is what is known as a context switch. Before a running process can be moved to a wait or blocked state all of the information about the process must be saved so that the process can resume later. This includes the state of the registers, file pointers, memory address points, and any other state information that is necessary for the process to to resume when it has access to a CPU again. A context switch is a rather expensive process and involves considerable resources. In order to maintain performance, the number of context switches should be minimized. If the number of context switches is not minimized, then conditions can arise where more time is spent on performing context switches than actually performing computation. Such a condition is referred to as "thrashing". A very simple example would be setting the time a process can spend running (called the quantum) to a very low value. Assuming that it took 1ms for a context switch and the quantum were set to 2ms, half of the time that each process was running would be spent performing context switches. In such conditions, performance is adversely impacted causing a decreasing level of throughput with an increasing response time.

1.3 Thesis Statement

This work examines the efficiency of particular scheduling algorithms in various configurations on several different types of workloads. To this end, the hypothesis for this research is:

An efficient virtual machine scheduling algorithm is important for increased throughput and decreased response time. Given varying workloads, there will be a particular scheduling algorithm that is more efficient at scheduling virtual machines for particular types of workloads. Thus, it is possible to fine tune the Xen hypervisor to maximize throughput and minimize response time with specific types of workloads.

1.4 Contributions of Work

This project develops and explores virtual machine workloads in order to evaluate the hypothesis by means of the following:

- Create three synthetic workloads that can be used in future research. The synthetic workloads represent a typical real world workload for Windows and Linux virtual machine environments.

- Measure the performance of each CPU scheduling algorithm for each case-study workload, using throughput and response time as metrics.

- Determine whether scheduling algorithm performance is workload specific and which scheduling algorithm is best suited for each of the synthetic workloads.

Specifically, this work develops a set of three synthetic workloads. The workloads are platform independent and can be run on any number of other hypervisors, such as those offered by Microsoft and VMware. Two of the case study workloads use open source software, allowing the entire collection of virtual machines to be released for use in future work. This is significant due to the fact that no such synthetic workloads currently exist.

The results of running a performance study with each workload and each scheduling algorithm provides insight into the impact of the hypervisor CPU scheduler on the overall performance of a virtual machine system.

1.5 Looking Ahead

This chapter has laid the introductory framework from which this research proceeded and provided a thesis statement. The issues associated with resource allocation, specifically related to the CPU, were outlined and a motivation given for why one would need to consider these issues. Now that the context for this research has been established, the remaining chapters will expand upon it.

Chapter 2 gives a review of related research to provide an assessment of the state of research in CPU scheduling and its relation to Xen. Points of interest are highlighted and their significance is discussed.

Chapter 3 gives a detailed architectural overview of the experimental system. The specifications of the physical hardware are discussed as well as details about the configuration of each of the three workloads.

Chapter 4 describes the design of the experiments. Care is given to discuss the rationale for each experiment and the steps taken to conduct them.

Chapter 5 presents the results of the performance study and discusses the significance of the findings. Concluding statements based on the research are also provided.

Chapter 6 provides suggestions for future work that could stem from this research. Emphasis is placed on areas where the work can be expanded and applied.

Chapter 2

Related Work

"The trouble with research is that it tells you what people were thinking about yesterday, not tomorrow. It's like driving a car using a rearview mirror."
—Bernard Loomis

This chapter details some key previous work that has laid the ground work for this research. Nearly every performance study on virtualization acknowledges the the importance of resource allocation via various scheduling algorithms [1] [2] [5] [8] [15]. Given this fact, it is surprising that there has not been more research conducted in the area of resource allocation in virtual environments.

2.1 Xen and the Art of Virtualization

Xen and the Art of Virtualization [1] marked the initial release of the Xen hypervisor in 2003. The paper discusses the architecture of the Xen hypervisor and provides details on a comprehensive performance study that was conducted to determine the cost of using virtualization. The authors found that there was a very small overhead of around 1%-5% associated with the use of Xen over a a series of benchmarks [1]. They also found that the performance of Xen was constantly higher than both User-Mode Linux and VMware Workstation 3.2, up to 20x better in some cases [1].

The performance study conducted by the researches was repeated and expanded by a group of graduate students at Clarkson University. They matched the hardware and benchmarks that were used in the original study and found that they were able to get results that were within 5% of the results obtained by Batham et al. [3].

2.2 Xen CPU Scheduling Research

Since the initial release of Xen there has been several papers published on hypervisor CPU scheduling algorithms and their relation to performance. They seek to fill the gap between the initial research does by the Xen team which was more of a justification for the use of Xen than a study on CPU scheduling. In general the follow-up research focuses on the balance of scheduling priority between domain. An overall theme is the use of both micro-benchmarks. While such methodologies allow for the discovery of overall trends they fail to give a real world notion of performance as this work seeks to do.

Cherkasova, et al. studied the effects of the borrowed virtual time (since removed from Xen), sEDF, and credit CPU schedulers on performance [2]. They used httperf to benchmark a web server along with two microbenchmark tools, `dd` for disk throughput and `iperf` for network throughput. The main focus was on the relationship between the CPU utilization of Domain0 and an additional VM that was running on throughput. They found that increasing the scheduling priority of Domain0 did not increase performance [2]. This result is significant, but the general methodology of using microbenchmarks does not translate into real world performance. This research expands on these results by using case study workloads.

Xu, et al. [15] conducted a performance study on the credit scheduler. They found that CPU bound jobs running in a VM have a negligible effect on I/O bound jobs running in a separate VM. The results were run with microbenchmarks that had the goal of pushing each test parameter, network or disk, to its limit. While there findings were interesting, these is little direct correlation to real world applications.

DeDiana [5] performed a study on the effects of various configurations of the sEDF CPU scheduler using multiple heterogeneous workloads. The main tools used for load generation were `dd` for disk throughput and `iperf` for network throughput. The results showed that the throughput of I/O applications is sensitive to CPU scheduling [5].

The overall trend seen in each work is the use of microbenchmarks. While they are good at providing a measure of the maximum amount of throughput that can be obtained by the network or disk, they fail to give any notion of the performance for a real world application load. Further, in each research study a quad-core system was not used. This work seeks to expand on the existing base research and study the effects of real world workloads. The real world workloads that were created for this study are outlined in the next chapter along with an overview of the virtual environment.

Chapter 3

Virtual Machine Architecture

"It would appear that we have reached the limits of what it is possible to achieve with computer technology, although one should be careful with such statements, as they tend to sound pretty silly in 5 years."
—John von Neumann, 1949

This chapter outlines the architecture of both the virtual machine environment and the physical resources used for testing. Architectural details and design decisions are included to provide an understanding of the underlying testing framework.

3.1 Hardware

To conduct this study parts were acquired to build a virtualization server. Major system components are outlined in Table 3.1. For the experimentation it was important that the server had more than adequate resources so that test results were not distorted by side effects like thrashing.

3.2 Workloads

Three synthetic workloads were created to facilitate testing the efficiency of the two Xen scheduling algorithms for each of the workloads. Efficiency was measured in

Table 3.1: Virtual System Specifications. An overview of the testbed system specifications.

Hardware	Specification
CPU	Intel Core 2 Quad Q9550 2.83Ghz with 12M L2 Cache and a 1333 Front Side Bus
Memory	8GB PC6400 DDR2 running at 800MHz
Primary OS Storage	80GB SATA 2 with 16MB Cache
VM Storage	1TB SATA 2 with 32MB Cache

terms of throughput (higher is better) and response time (lower is better). The measurements were taken at the virtual machine level using benchmarking tools specific to each workload and at the hypervisor level using the `Xenmon` tool [8]. `Xenmon` provides statistics about how much time each domain is using a CPU, blocked on an I/O event, and waiting on the ready queue [8]. Together these three measurements account for the time a virtual machine is running. In terms of the metric of response time, an efficient scheduler is defined as one that minimizes the amount of time each VM is in the hypervisor wait queue. XenMon also reports I/O count which is a rough measure of I/O requested by a domain [8]. Through the I/O count statistic a measure of throughput can be captured as a higher I/O count of a particular scheduler configuration is indicative of a higher level of throughput.

Each of the workloads that were created will be discussed in the following sections.

3.2.1 Web Application

The web application workload was created to replicate a very simple web application that uses MySQL, Apache, and NFS (Network File System) servers. Each server

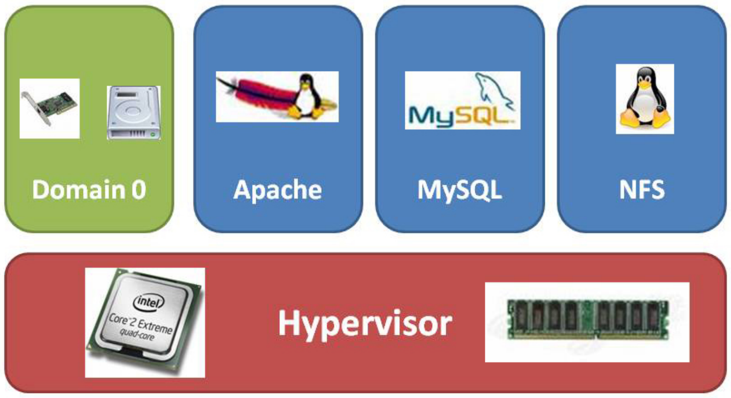


Figure 3.1: Web Application Workload Architecture. A high-level view of the web application workload.

VM is separate so there is transfer of data between multiple virtual machines as they work together to serve web content, which represents a layer of isolation that is commonplace in many large scale web applications. The actual application is written in PHP (PHP: Hypertext Preprocessor) and performs a number of representative SQL (Structured Query Language) queries. The number of clients was simulated using JMeter[7] so differing numbers of users interactions could be tested. The virtual machine architecture is depicted in Figure 3.1.

3.2.2 VPS Application

The virtual private server (VPS) hosting workload was created to simulate a workload that is present in virtual shared hosting environments. In such environments, each user is given root access to a virtual machine in which any number of installed applications can be run. Each user's VPS instance runs on a shared hardware node which means that the hypervisor's CPU scheduler must give a fair share of the CPU to each VM to maintain overall performance. For this workload, each VPS was generalized as a single instance of Apache and MySQL. This is a typical default Linux server set-up

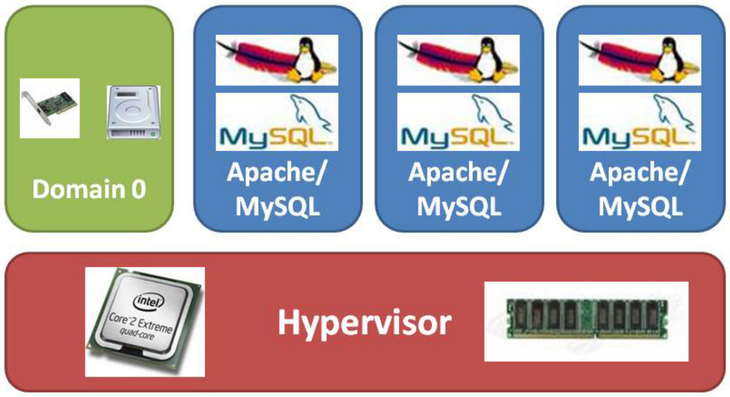


Figure 3.2: VPS Hosting Workload Architecture. A high-level view of the VPS application workload.

refereed to as a LAMP (Linux, Apache, MySQL, PHP) installation. Each VPS ran the same web application from the web application workload. The virtual machine architecture is depicted in Figure 3.2.

3.2.3 Windows Domain

The Windows Domain case study workload is designed to replicate a simple corporate Windows domain consisting of an Active Directory Server, an MSSQL Server, an Exchange Server, and a SharePoint Portal Server. This workload uses all closed source tools. It is included because it represents the set of servers that are run in many businesses and are prime candidates for virtualization. Further, both Citrix and Sun have adopted the Xen hypervisor in their virtualization products which are both aimed at datacenter usage to consolidate servers, many of which are running Windows Server operating systems. The virtual machine architecture of the workload is depicted in Figure 3.3.

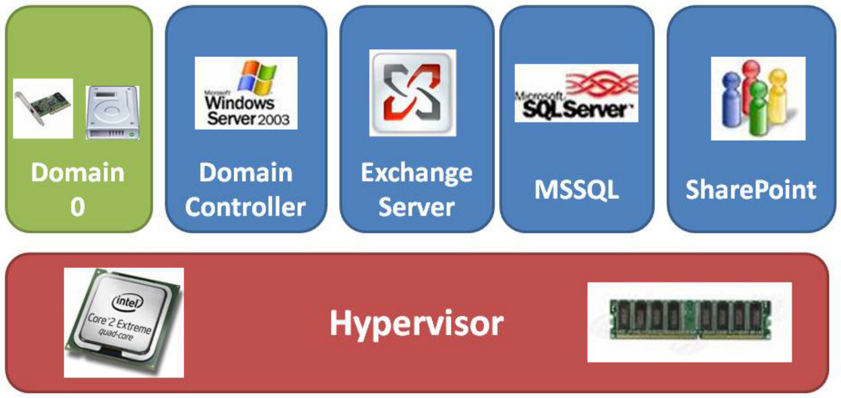


Figure 3.3: Windows Domain Workload Architecture. A high-level view of the windows domain application workload.

3.2.3.1 Active Directory

Active Directory (AD) is Microsoft's authentication and user management system. AD consists of one or more servers running Server 2003 with the domain controller role installed. After a domain is created, machines are joined to it and authenticate though one of the domain controllers. In a typical organization with many users and computers, AD provides user authentication. Exchange and SharePoint also use AD to authenticate users for access to email and portal services.

3.2.3.2 Exchange

Exchange is Microsoft's email solution that has integrated contact and calendar management. It provides web mail, MAPI (Messaging Application Programming Interface), POP3 (Post Office Protocol), SMTP (Simple Mail Transfer Protocol), and IMAP (Internet Message Access Protocol). The Outlook mail client provides connectivity to a user's mail though the use of MAPI and RPC (Remote Procedure Protocol).

3.2.3.3 SharePoint

Microsoft Office SharePoint Server (MOSS) is a collaberative web-based portal system that allows users to upload and share documents. Many businesses and organizations leverage MOSS to provide employees with a central file repository and to allow for easy collaboration. MOSS runs on its own instance of Server 2003. It heavily relies on Microsoft SQL (MSSQL) for storage of configuration files, user content, and documents. Documents are stored in the database as binary large object (blob) datatypes.

3.2.3.4 MSSQL

Microsoft SQL (MSSQL) Server is Microsoft's implementation of a SQL (Structured Query Language) database server. MSSQL is used to store all of the SharePoint configuration, user content, and user uploaded documents.

3.3 Test Machine

All of the load generation was conducted from a Windows XP SP3 computer, herein referred to as the test machine. The test machine was joined to the Windows domain that was established as part of the Domain workload to allow for the proper functioning of the MOSS and Exchange test tools. The hardware specifications of the test machine can be seen in Table 3.2.

3.4 Network Overview

The network architecture was designed so that there was a private network between the test machine, VMs, and the Xen server. The private network consisted of a crossover Ethernet cable between the Xen server and the test machine. On the Xen server the physical Ethernet adapter, eth0, is bound to a virtual Ethernet adapter

Table 3.2: Test Computer Specifications. An overview of the test computer specifications.

Hardware	Specification
CPU	Intel Pentium E2180 2.0Ghz Dual Core with 1M L2 Cache and an 800 Front Side Bus
Memory	4GB PC5300 DDR2 running at 400MHz
Storage	80GB SATA 2 with 16MB Cache

called peth0. The virtual adapter, peth0, is connected to a bridge interface. Each of the VMs has a virtual Ethernet adapter on the bridge interface. In this manner each VM is presented the equivalent of a direct connection to the private network. In addition to the private network, a public network with no connectivity to the VMs was included to allow for management. The network architecture is depicted in Figure 3.4.

This chapter provided an overview of the test system, the Xen server, and each case study workload application. The following chapter will discuss the experimental design that was built upon this architecture.

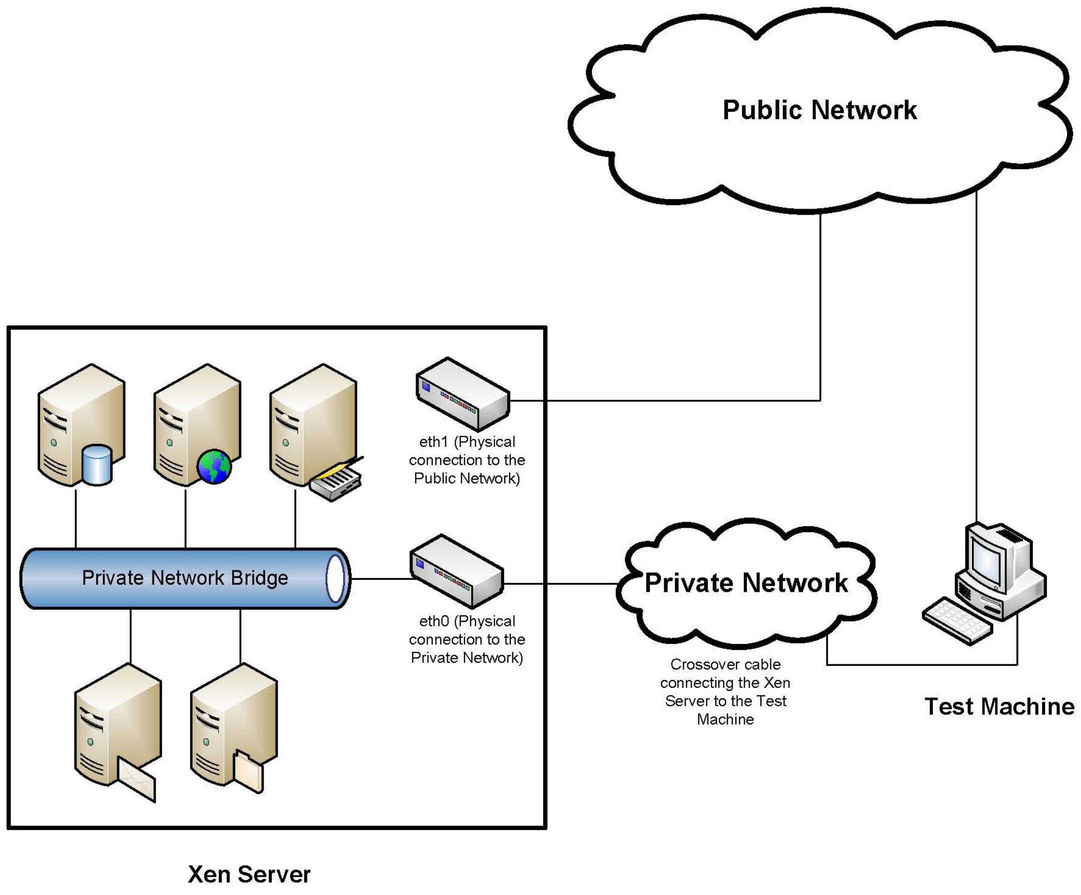

Public Network
eth1 (Physical connection to the Public Network)
Private Network Bridge
eth0 (Physical connection to the Private Network)
Private Network
Crossover cable connecting the Xen Server to the Test Machine
Test Machine
Xen Server

Figure 3.4: Network Architecture. An overview of the network architecture created for experimentation.

Chapter 4

Experiment Design

"Beware of bugs in the above code; I have only proved it correct, not tried it"

—Donald Knuth

In Chapter 3 the physical and virtual architecture of the test system were outlined. This chapter builds upon that discussion by providing an overview of the experimental design. The overall goal of each series of experiments was to introduce a varying user load on each case study application workload so that the efficiency of each CPU scheduling algorithm could be determined, both at the application level and the virtual machine level.

4.1 Web Application Workload Experiment

The web application workload focused on a web application that was created to represent a dynamic website where a user's visit to the page involved multiple interactions with a database. The throughput and response time of the application was measured with JMeter.

With each of the experiments, five configurations of the credit scheduler and five configurations of the sEDF scheduler were used. Each configuration was set to give a varying level of priority to the scheduling of the main VMs involved in the test. The credit scheduler was used with weights of 256, 512, 1024, 2048, and 4096 set for the NFS, MySQL, and Apache Server VMs. The configurations for the sEDF scheduler are outline in Table 4.1.

Table 4.1: Web Application sEDF Scheduler Configurations. An overview of the slice and period used for the sEDF scheduler. The value in parenthesis indicates the name used to refer to that scheduler configuration.

Configuration (name)	Domain0 Slice (ms)	MySQL, NFS, Apache VM Slice (ms)	Period (ms)
Default (sedf-20)	15 (with 20ms period)	20	100
Equal Slices (sedf-2.5)	2.5	2.5	10
More Weight to Main VMs (sedf-2.8)	1.6	2.8	10
Equal Slices w/ Large Period (sedf-250)	250	250	1000
Greater Main VMs Slices w/ Large Period (sedf-280)	160	280	1000

4.1.1 Application Design

The web application was designed to represent a simple dynamic webpage that loads information from a database. This type of interaction is very common in the design of modern websites where content changes rapidly and is often stored in a central database. The basic design of the web application is outline is Figure 4.1. The main page was written in php and relies on a MySQL database. The web application performs several select statements and an insert statement to a table in the database to simulate the process of a logging a user's visit and a user's interaction with the application.

```
Send webpage header to client
Simulate a client login
Set session variables
Log a user's visit to the site
Get main html from database
Send webpage footer html to client
```

Figure 4.1: Pseudo Code for the Web Application. A basic outline of the web application design.

4.1.2 Running the Experiment

Each experiment was run using a Widows PowerShell script. PowerShell was used for its ability to bring Linux-like scripting functionality to Windows. The pseudo code for the script is shown in Figure 4.2

```
for each scheduling algorithm configuration
        for each user load
                run five times
                        run SQL setup script
                        run JMeter in the background
                        run xenmon for 30 minutes
                        run SQL teardown script
```

Figure 4.2: Pseudo Code for the Web Application Workload Testing Script. A basic outline of the script used to run the web application workload experiment.

4.2 VPS Workload Experiment

The VPS workload used the same application that was designed for the web application workload. The application was installed on each instance of a VPS. Apache JMeter was used to create a user load of 25, 50, 100, 500, and 800 simultaneous users on each VPS.

4.2.1 Running the Experiment

The experimentation process was scripted using Windows PowerShell. The pseudo code of the script is provided in Figure 4.3. The throughput and response time of the application on each VPS was measured with Apache JMeter.

```
for each scheduling algorithm configuration
        for each user load
                run five times
                        run SQL setup on each VPS
                        run JMeter on each VPS
                        run xenmon for 30 minutes
                        run SQL teardown script on each VPS
```

Figure 4.3: Pseudo Code for the VPS Workload Testing Script. A basic outline of the script used to run the VPS workload experiment.

4.3 Windows Domain Workload Experiment

The primary services that were focused on for the Windows domain workload were Exchange and MOSS (see Section 3.2.3.2 and Section 3.2.3.3 for more details on each).

Exchange performance was evaluated with a Microsoft tool called Exchange Load Generator (LoadGen). LoadGen creates a customizable user load on an Exchange server [9]. More details about the tool may be found in Appendix B.2. For the experimentation, user loads of 25, 50, 100, 500, and 1000 were used. Each user was set to a heavy level of activity to represent very active Exchange users, that is, users that are constantly sending and receiving emails and creating multiple calendar events.

Microsoft does not have an analogous tool to test MOSS, so Apache JMeter was used. JMeter allows web events to be captured though the use of a proxy server (see

Appendix B.1 for more details). A proxy server is an intermediary server that resides between a client's web browser an any network resources that reside on a network. When the proxy server is initialized, every web request from a browser goes through the JMeter proxy server and creates a web event that can then be put into a test suite and played back. For the experimentation, a test suite was created that involved going to the main MOSS site, creating a table, and then uploading a small document. User loads of 25, 50, 100, and 500 were tested.

With each of the experiments, four configurations of the credit scheduler and four configurations of the sEDF scheduler were used (except for MOSS where an additional configuration of the credit scheduler was used). Each configuration was set to give a varying level of priority to the scheduling of the main VMs involved in the test. In the Exchange tests using the credit scheduler, the Exchange VM was given weights of 256, 512, 1024, and 2048. For the MOSS test the same weights for the credit scheduler were used, but the weights were set on both the MOSS VM and the MSSQL VM, as both VMs are integral to the operation of a sharepoint site. Both the MOSS and Exchange tests used the same configuration values for the sEDF scheduler. The sEDF configurations are outline in Table 4.2.

4.3.1 Running the Experiments

Each experiment was run using a PowerShell script. The pseudo code for the script is provided in Figure 4.4. The PowerShell script performed the setup of each test tool, started the test tool, and ran `xenmon`.

Table 4.2: Windows Domain sEDF Scheduler Configurations. An overview of the slice and period used for the sEDF scheduler.

Configuration (name)	Domain0 Slice (ms)	Main VM Slice (ms)	Other VM Slice (ms)	Period (ms)
Default (sedf-20)	15 (with 20ms period)	20	20	100
Equal Slices (sedf-2)	2	2	2	10
More Weight to Main VMs (sedf-2.8)	2	2.8	1.8	10
Equal Slices w/ Large Period (sedf-200)	200	200	200	1000
Equal Slices w/ Large Period (sedf-175)	175	175	175	1000

```
for each user load
        for each scheduling algorithm configuration
                run five times
                        run SQL setup script
                        start the testing tool in a background process
                        run xenmon for 30 minutes
                        run SQL teardown script
```

Figure 4.4: Pseudo Code for the Windows Workload Testing Script. A basic outline of the Windows PowerShell scripts that were used for the Window domain workload experiments.

4.4 Results Analysis

The experimental design resulted in the generation of a large about of data. Each script produced 125 output files on the test machine and 500 files on the Xen server. This section describes the process used to reduce all of the collected data into a manageable format.

4.4.1 JMeter

All of the JMeter tests were configured to output the results to an XML (eXensible Markup Language) file with a `jtl` extension. The output files contained information about every request that was made. An example of an entry in the output file is provided in Figure 4.5 with an explanation of each logging parameter outlined in Table 4.3.

```
 <httpSample t="165" lt="165" ts="1269905827968" s="true"
lb="HTTP Request" rc="200" rm="OK" tn="Thread Group 1-3"
dt="text" by="5017"/>
```

Figure 4.5: JMeter Log Entry. A sample of a JMeter log entry. An explanation of each field is provided in Table 4.3.

The log files were quite large since they were in XML format and there were many requests. To help prune the information, each logfile was processed by a python script that parsed the jtl files and output csv (comma separated value) files that contained a single column with each response time. For example, the script would take the sample entry in Figure 4.5 and return 165 on single line.

Once the log files were in the more portable csv format, they were imported into Microsoft Excel. A sheet was created for each userload. Within each sheet a column was created for each run of each scheduler configuration. The csv files containing the response times of each configuration were then imported into the appropriate columns. After all of the data was imported into Excel, a summary sheet with a condensed view of the data was created. The summary sheet contained average response times, average throughput, the standard deviation of each averaged value, and a 95% confidence interval calculation. Using this summary data, graphs were then created.

Table 4.3: JMeter Log Entry Parameter. A description of each of the JMeter parameters captured.

Parameter	Meaning
httpSample	Specifies that the entry is an HTTP request
t	Response time
lt	Latency
ts	Timestamp of request
s	True/False value that specifies whether the request was successful
lb	Label of the test
rc	Response code
rm	Response message
tn	Thread name
dt	Data type
by	Bytes transferred

4.4.2 LoadGen

The LodGen tool produced an XML file that contained a summary of the latencies for each task completed, such as logging on, creating a calendar event, sending email, etc. This data was already reduced to a point that was manageable. The latencies of several tasks were chosen and used to present the performance of each scheduler configuration. Unlike the JMeter tool, LoadGen does not attempt to perform as many connections as possible so a measure of throughput was not obtainable.

4.4.3 Xenmon

Xenmon was used to collect CPU allocation information about each VM at the hypervisor level, namely the amount of time that each domain spent in the blocked, running, and ready states. More information on the tool can be found in Appendix B.7. Xenmon was configured to output all of its logging information to a file at 1 second intervals and run for a total 1800 seconds (30 minutes). By default xenmon collects information for each domain and saves it to a file with a .log extension. Since the log files were tab delimited they lent themselves to easy importation into an R dataframe. To simplify the process, an R function was written that took the location of a log file, the name of a log file, and a domain number. The function then returned the average percentages that the given domain spent in the blocked, running, and wait states. The averages were taken over the 5 test runs that were performed for each combination of scheduler configuration and user load.

This chapter presented the experimental design that was used to conduct all of the experiments used in this research. Now that a solid background and rationale to the execution of the experimentation has been given, the results can be presented. In the next chapter the results of the experimentation are presented along with an analysis of the trends.

Chapter 5

The Results: Analysis and Conclusions

"It is hardware that makes a machine fast. It is software that makes a fast machine slow."
—Craig Bruce

This chapter presents the results obtained from running the experiments described in the previous chapter. An analysis of the results for each application and scheduling configuration pair is given along with an explanation and consideration of the significance and limitations. This is followed by a summary and conclusions based on the results collected.

5.1 Introduction

As expected, the performance study experiments produced a large amount of data. This data was reduced using techniques that were outlined in Chapter 4, namely through the use of python and R scripts, along with Microsoft Excel. In addition to the reduction, a statistical analysis was performed on the data. This analysis included the determination of standard deviations and 95% confidence intervals.

Standard deviation is a measure of dispersion, i.e., how close a set of values is to the average value of the data set. Higher values of standard deviation indicate a greater variability in the results. The general formula for calculating standard deviation is given in Equation 5.1. The standard deviation, σ, is equal to the square root of the summation of each value, x, subtracted from the mean value, $\bar{x}$, squared divided by the total number of items in the data set, N. Using this calculation, it was possible to determine the variability of the average values for throughput and response time that were calculated.

$$\sigma = \sqrt{\frac{\Sigma(x - \bar{x})^2}{N}} \tag{5.1}$$

The value of standard deviation only indicates the degree of variability within a set of data. However when coupled with a confidence interval, a greater understanding of the nature of variability of the results can be obtained. A confidence interval is a measure of the variability of a set of data within a given percentage of the data. For example, a 95% confidence interval of ± 10 indicates that 95% of the values in the data set are within a range of $[\bar{x} - 10, \bar{x} + 10]$, where $\bar{x}$ is the mean of the values in the data set. Smaller confidence intervals indicate that that there is less variability of the majority of the data. The general formula for calculating a 95% confidence interval is given in Equation 5.2 where N is the number of values in the data set, $\bar{x}$ is the mean of the data set, and σ is the standard deviation of the data set. Using this calculation it was possible to determine a range that 95% of the data results fell within and helped to establish a level of confidence in the mean values that were calculated.

$$CI = [\bar{x} - 1.96(\frac{\sigma}{\sqrt{N}}), \bar{x} + 1.96(\frac{\sigma}{\sqrt{N}})] \tag{5.2}$$

The remaining sections of this chapter use these statistical techniques to analyze the results. A discussion of the rationale behind the observed trends and their significance is included with the analysis.

5.2 Web Application Results

The web application study showed that results for throughput and response time varied with both the user load and scheduler configuration. An overall trend that emerged was that there was not one scheduler that performed best over all of the user loads. However, for each user load, there were one or two scheduler configurations that maximized throughput and minimized response time. These values are indicated by bold typeface in Tables 5.1 and 5.4. For example, the credit-256 and sedf-2.5 scheduler configurations had the highest throughput for 25 users with values of 127.57 and 126.86, respectively.

During the execution of each test `xenmon` ran on Domain0 to collect CPU scheduling information about the blocked, waiting, and cpu execution time for each virtual machine. The output that was captured provided insight to the characteristics of each VM across user loads. A summary of the output for the web application VMs is provided in Tables 5.7, 5.8, 5.9, and 5.10. The Apache VM `xenmon` output indicates that the VM was CPU bound. The data shows that the VM spent approximately 90% of its execution time in a waiting state while spending only around 1% of its execution time in a blocked state (as seen in Table 5.7). The NFS VM was both I/O and CPU bound, although more so CPU bound. This is due to the fact that this VM was performing mostly I/O in the form of disk and network access as it fulfilled its role as a file server. The VM spent approximately 40% of its execution time blocked and around 60% of its execution time waiting for access to a CPU as seen in in Table 5.8. The MySQL VM had similar constraints to the NFS VM, in that it was CPU and I/O bound as shown in Table 5.9. However, the VM spent less time blocked and and nearly double the amount of time using the CPU (3%-8%), as seen in Table 5.9. An interesting trend that was seen with all of the VMs was that the CPU scheduling algorithm did not have a marked effect on the amount time a VM spent in each state. The `xenmon` output also revealed that for each user load, each scheduling configuration was close with respect to blocked, run, and wait time percentages.

An overall correlation can be seen between scheduler configurations in Figure 5.1 and Figure 5.2. Configurations that had high levels of throughput tended to have low levels of response time, independent of user load and scheduler configuration. This type of correlation is expected given the nature of the web application workload. If each connection has a lower response time, then more connections can occur in any given interval of time. This is especially true when the confidence interval has a smaller range. For example, with 25 users the sedf-2.5 configuration has a small confidence interval for response, as seen in Figure 5.2, with a high level of throughput, as seen in Figure 5.1.

Another trend that emerged with response time was a jump in response time between 100 and 500 users (as seen in Figure 5.2). Increasing the user load from 25 to 50 users caused, in general, a doubling of response time over each of the scheduler configurations. This doubling was also seen when the user load was increased from 50 to 100 users. However, an increase in user load from 100 to 500 users caused a much more significant increase in response time. These increases are shown graphically in Figure 5.2 and are also represented in Table 5.4. Along with increasing response times came increasing confidence intervals, indicating a greater variability in the response time from request to request (as seen in Table 5.6).

An interesting trend was seen with the sEDF scheduler when the slice and period were set to give a higher scheduling preference to Domain0. In each instance (sedf-20 and sedf-280) neither throughput nor response time were optimized. This trend is interesting because it seems to be intuitive that giving a higher scheduling preference to Domain0 would increase throughput due to the fact that all I/O must pass through Domain0. However, it turned out that more fairly sharing the CPU resources between all of the VMs leads to higher level of throughput and lower response time. For example, this can be seen in in Table 5.1 where with a user load of 25 users the sedf-2.5 configuration was able to achieve a higher level of throughput than the sedf-20 configuration, 126.86 and 87.26 requests per second, respectively. Coincidentally the sedf-20 configuration is the default configuration of the sEDF scheduler. Therefore, this analysis would indicate that the default configuration of the sEDF scheduler should be changed to maximize throughput.

Another trend observed is related to the performance of the credit-256 scheduler configuration, which is the default scheduler configuration for the Xen hypervisor. This configuration performed best only for user loads of 25 and 100 users. However, in each case it was very close, if not indistinguishable, from the next best scheduler configuration. For example, this can be seen in Table 5.1 under the 25 users column with the credit-256 and sedf-2.5 scheduler configurations. In fact, the credit scheduler was only the clearly optimal scheduler configuration for the tests with a user load 50 and 100 users.

A recurring conclusion is that a simple increase in scheduling priority did not correspond to an optimal scheduler configuration. Rather, there are optimal configurations of each scheduler for each user load. This points to a distinct benefit from fine tuning the CPU scheduling algorithm based on the anticipated user load of a server running a web application. Failure to do so could result in suboptimal throughput with a high response time, especially as the user load on a server increases.

Table 5.1: Web Application Throughput Results. An overview of the web application throughput results.

	25 Users	50 Users	100 Users	500 Users	800 Users
credit-256	**127.57**	90.97	**94.39**	66.15	89.23
credit-512	90.30	101.65	74.39	79.36	71.37
credit-1024	118.79	77.72	**91.86**	83.97	82.07
credit-2048	108.26	**107.08**	77.30	**95.42**	83.30
credit-4096	115.04	91.07	89.46	70.24	77.03
sedf-20	87.26	71.87	84.88	70.67	91.37
sedf-2.5	**126.86**	85.84	76.45	82.42	66.86
sedf-2.8	85.84	71.41	70.72	80.80	**102.60**
sedf-250	104.52	101.06	89.75	**94.36**	83.54
sedf-280	103.01	84.19	78.71	69.35	79.88

Table 5.2: Web Application Throughput Standard Deviation. An overview of the standard deviation of the throughput values obtained in the web application tests.

	25 Users	50 Users	100 Users	500 Users	800 Users
credit-256	28.83	25.05	26.14	9.76	26.71
credit-512	7.95	21.57	4.81	24.57	8.35
credit-1024	33.20	6.88	36.91	23.81	23.67
credit-2048	34.12	26.34	14.65	23.80	17.15
credit-4096	42.49	25.83	29.25	6.84	19.43
sedf-20	10.93	11.10	17.59	10.83	21.85
sedf-2.5	24.05	27.13	17.74	23.39	13.54
sedf-2.8	19.98	18.97	17.40	28.48	23.06
sedf-250	21.74	16.14	26.67	27.26	25.03
sedf-280	28.75	28.92	26.64	15.57	20.09

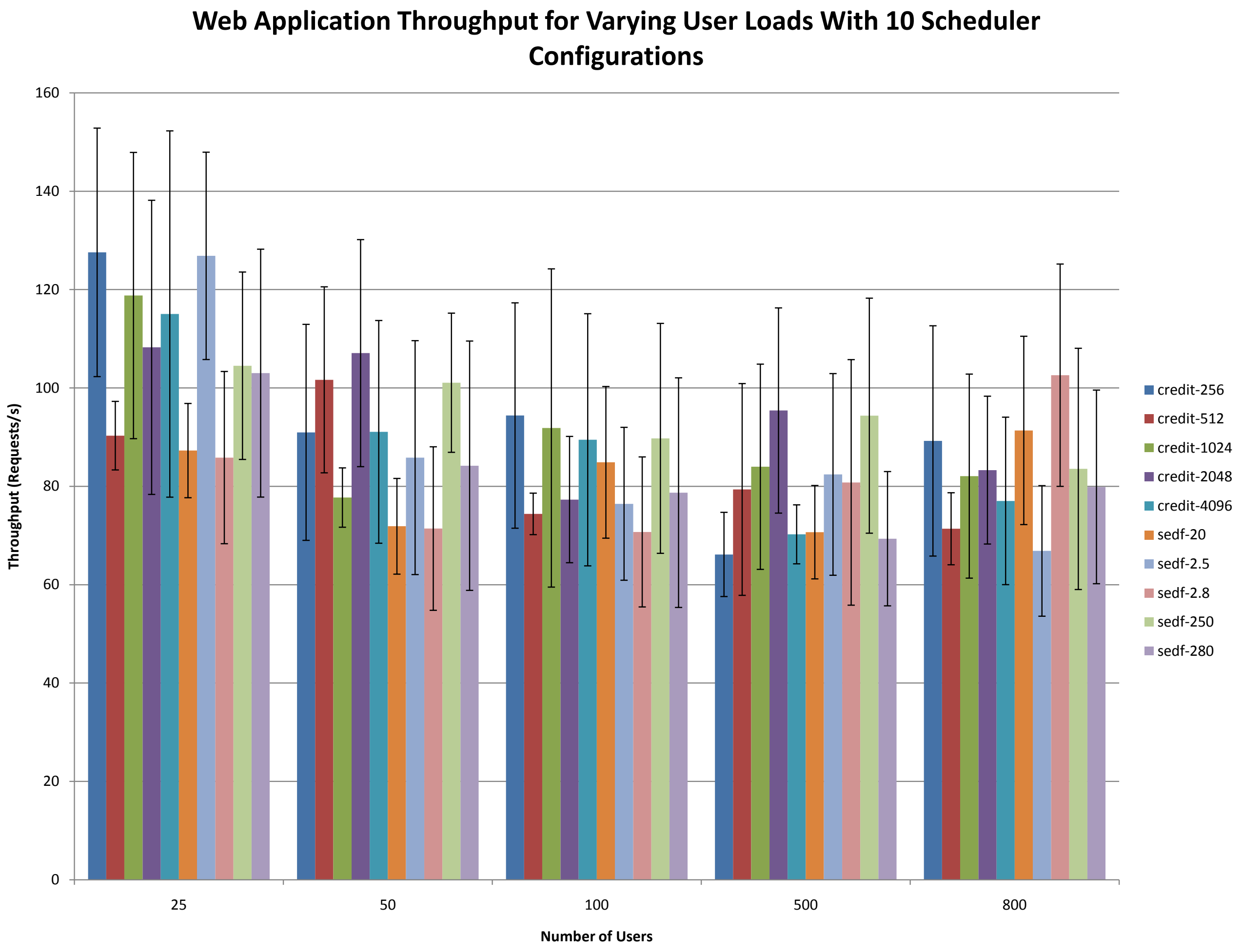


Figure 5.1: Web Application Throughput Graph. A graphical depiction of the average throughput for each scheduler configuration under each user load with 95% confidence intervals.

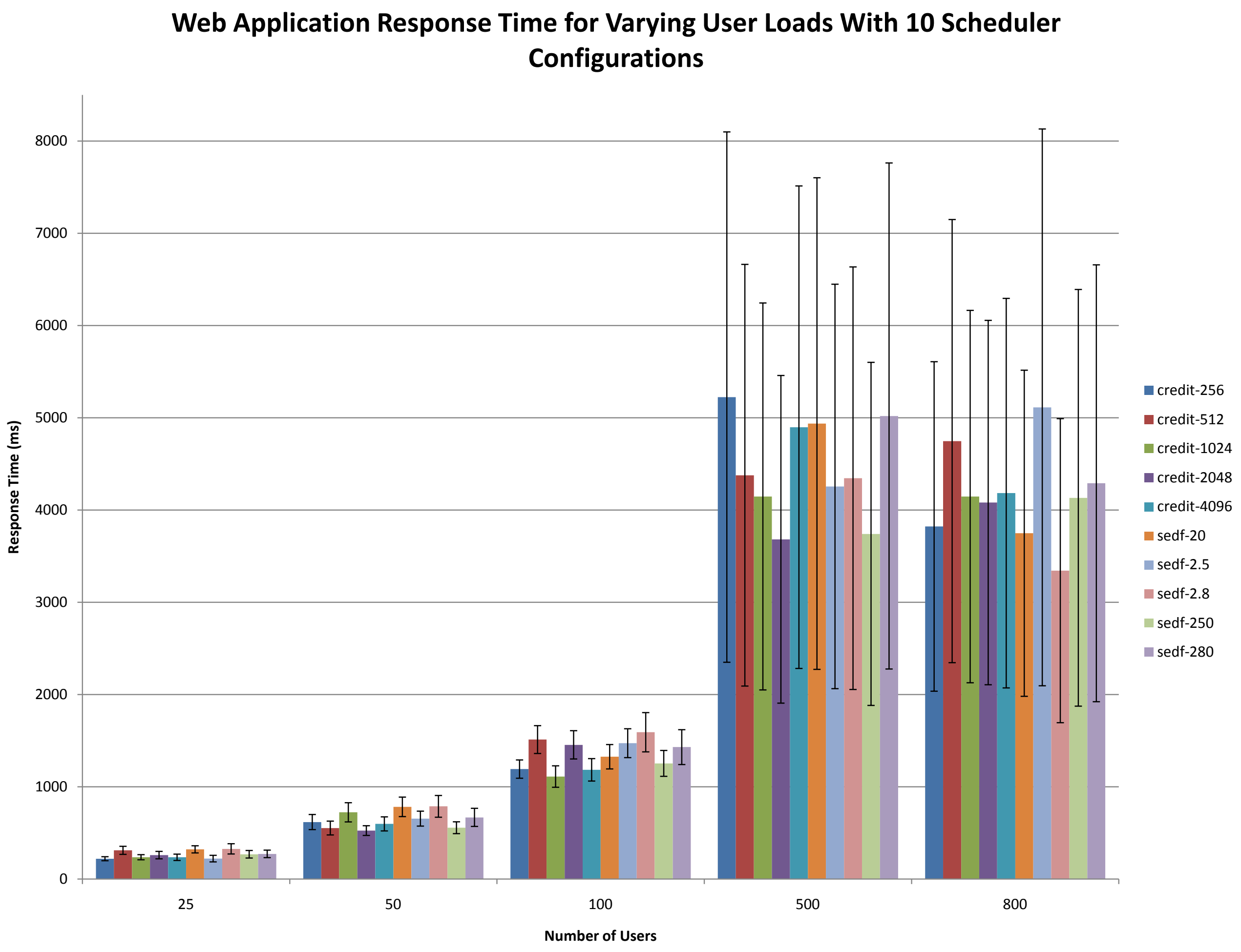


Figure 5.2: Web Application Response Time Graph. A graphical depiction of the average response time for each scheduler configuration under each user load with 95% confidence intervals.

Table 5.3: Web Application Throughput 95% Confidence Intervals. An overview of the 95% confidence interval of the results obtained in the web application tests. Each value indicated in the table is a ± value from the mean.

	25 Users	50 Users	100 Users	500 Users	800 Users
credit-256	25.27	21.96	22.91	8.56	23.41
credit-512	6.97	18.91	4.22	21.53	7.32
credit-1024	29.10	6.03	32.35	20.87	20.75
credit-2048	29.91	23.08	12.84	20.86	15.03
credit-4096	37.24	22.64	25.63	6.00	17.03
sedf-20	9.58	9.73	15.42	9.49	19.15
sedf-2.5	21.08	23.78	15.55	20.50	13.27
sedf-2.8	17.51	16.63	15.25	24.97	22.60
sedf-250	19.06	14.15	23.38	23.90	24.53
sedf-280	25.20	25.35	23.35	13.65	19.69

Table 5.4: Web Application Response Time Results. An overview of the web application response time results.

	25 Users	50 Users	100 Users	500 Users	800 Users
credit-256	**220.30**	618.06	**1192.09**	5224.85	3822.37
credit-512	311.22	553.07	1512.10	4377.85	4747.73
credit-1024	236.58	723.49	**1111.19**	4147.77	4146.33
credit-2048	259.60	**525.53**	1455.03	**3683.12**	4081.55
credit-4096	235.78	597.93	1184.03	4898.51	4183.56
sedf-20	322.05	782.65	1325.95	4937.94	3748.72
sedf-2.5	**221.64**	655.09	1472.74	4256.22	5113.35
sedf-2.8	327.41	788.07	1591.51	4345.55	**3343.20**
sedf-250	268.86	556.45	1253.92	3741.93	4132.89
sedf-280	272.78	668.55	1430.21	5020.05	4290.56

Table 5.5: Web Application Response Time Standard Deviation. An overview of the standard deviation of the response times obtained in the web application tests.

	25 Users	50 Users	100 Users	500 Users	800 Users
credit-256	279.20	889.60	1097.09	26644.68	19244.45
credit-512	472.89	861.67	1483.54	23206.43	23122.98
credit-1024	343.43	1046.29	1271.06	21905.31	20851.13
credit-2048	477.62	621.45	1535.92	19790.31	20554.25
credit-4096	420.28	829.48	1310.98	25000.24	21138.53
sedf-20	413.43	1022.37	1389.26	25544.79	19257.95
sedf-2.5	457.66	855.04	1560.97	22704.97	25161.39
sedf-2.8	587.43	1136.45	2039.60	23459.66	17026.94
sedf-250	476.90	735.39	1514.84	20593.05	21059.83
sedf-280	472.08	1027.21	1910.56	26032.41	21579.15

Table 5.6: Web Application Response Time 95% Confidence Intervals. An overview of the 95% confidence interval of the response times obtained in the web application tests. Each value indicated in the table is a ± value from the mean.

	25 Users	50 Users	100 Users	500 Users	800 Users
credit-256	21.68	81.83	99.08	2874.76	1786.019
credit-512	43.64	74.93	150.96	2285.65	2401.97
credit-1024	27.64	104.11	116.28	2097.44	2018.3
credit-2048	40.25	52.66	153.22	1775.99	1975.16
credit-4096	34.35	76.22	121.53	2615.40	2111.51
sedf-20	38.81	105.76	132.24	2664.79	1767.57
sedf-2.5	35.62	80.91	156.54	2192.40	3018.05
sedf-2.8	55.59	117.89	212.77	2290.43	1648.13
sedf-250	40.91	64.14	140.27	1859.77	2258.55
sedf-280	40.77	98.24	188.89	2742.99	2368.03

Table 5.7: Web Application Apache VM CPU Time. An summary of the percentage of total runtime that the Apache VM spent in the blocked, running, and waiting states.

	25 Users			50 Users			100 Users			500 Users			800 Users		
	block	run	wait	block	run	wait	block	run	wait	block	run	wait	block	run	wait
credit-256	0.1	10	90	1.1	9.8	89.1	0.1	12.4	87.5	0.4	13.8	85.8	0.1	12.7	87.1
credit-512	1.2	10.4	88.6	1.2	10.4	88.4	1.4	11.6	87.1	0.6	12.9	86.5	1	12.3	86.7
credit-1024	1.4	10	88.6	0.4	10.9	88.8	2.2	11.1	86.6	0.2	13.6	86	0.5	13.6	85.9
credit-2048	1.4	9.9	88.7	0.7	10.6	88.7	1.2	12.4	86.4	0.8	12	87.1	0.6	12.2	87.3
credit-4096	1.3	9.9	88.8	1.2	11.6	87.3	1.3	12	86.7	0.4	13.2	86.3	0.6	13.3	86.1
sedf-20	0.3	7.7	92.1	1.8	8.1	90.1	0.6	11.7	87.7	0.6	11.6	87.8	0.1	13.5	86.4
sedf-2.5	0.6	7.3	92.1	1.5	7.9	90.6	0.2	9.9	90	0.2	9.7	90.1	0.1	7.9	92
sedf-2.8	2	6.8	91.2	1.6	8.1	90.3	0.7	11.6	87.7	0.3	10.3	89.4	0.1	9	90.9
sedf-250	2.1	8.2	89.8	1.8	8.5	89.7	1	11.1	89.3	0.2	10.5	89.3	0.8	13.1	86.1
sedf-280	12	7.7	90.9	1.6	9.6	88.8	0.8	10.1	89.1	0.4	11.2	88.4	0.4	11.2	88.5

Table 5.8: Web Application NFS VM CPU Time. An summary of the percentage of total runtime that the NFS VM spent in the blocked, running, and waiting states.

	25 Users			50 Users			100 Users			500 Users			800 Users		
	block	run	wait	block	run	wait	block	run	wait	block	run	wait	block	run	wait
credit-256	36.1	2.1	61.8	40	1.3	58.8	37.6	1.7	60.7	39.6	1.3	59.1	37.8	1.6	60.6
credit-512	37.2	1.7	61.1	37.5	1.7	60.7	38.4	1.5	60.1	39.9	1.2	58.8	38.6	1.4	60
credit-1024	37.6	1.7	60.7	36.4	1.8	61.8	39.9	1.3	58.8	40.7	1.3	58.1	40	1.2	58.8
credit-2048	38.4	1.6	60.1	38.5	1.6	59.9	39.6	1.3	59.1	38.6	1.5	60	39.5	1.5	59
credit-4096	37	1.6	61.1	38.5	1.5	60	39.6	1.3	59	40.1	1.3	58.6	40	1.3	58.6
sedf-20	43.2	1.4	55.4	41.6	0.8	57.6	39.9	1.1	59.1	46.3	0.8	52.9	41.4	0.7	57.9
sedf-2.5	42.9	1.3	55.8	46.2	0.8	53	45.1	0.9	54	41.8	1.1	57.1	39	1.1	59.9
sedf-2.8	45	0.9	54.2	42.2	1	56.8	46.3	0.8	52.9	40.5	1	58.5	40.3	1.1	58.6
sedf-250	41.7	1.1	57.1	39.5	1.1	59.5	42.3	1	56.7	44.2	0.7	55.1	45.8	1.1	53.1
sedf-280	42.1	1.5	56.4	40.9	1.2	57.9	45.8	0.7	53.5	43.4	0.8	55.8	45.8	1.2	53

Table 5.9: Web Application MySQL VM CPU Time. An summary of the percentage of total runtime that the MySQL VM spent in the blocked, running, and waiting states.

	25 Users			50 Users			100 Users			500 Users			800 Users		
	block	run	wait	block	run	wait	block	run	wait	block	run	wait	block	run	wait
credit-256	25.8	7.6	66.5	34.2	4.6	61.2	29.7	6.2	64.1	33.8	4.8	61.4	30.6	5.9	63.5
credit-512	29.4	6.1	64.4	29.2	6.3	64.6	31.4	5.4	63.1	34.6	4.5	60.9	32.1	5.3	62.6
credit-1024	29.8	6.1	64.1	28.3	6.8	64.9	33.8	4.7	61.5	34.7	4.6	60.7	34.7	4.4	60.9
credit-2048	31.4	5.7	63	30.5	6	63.5	33.9	4.5	61.5	31.6	5.5	62.9	32.6	5.3	62.1
credit-4096	29	6.3	64.6	31.8	5.3	62.9	33.3	4.63	62	34	4.8	61.2	34.8	4.4	60.8
sedf-20	25.4	6.4	68.2	39.7	3.3	57	30.5	5.1	64.4	33.4	3.5	63.2	34.9	3.8	61.3
sedf-2.5	28.9	5.4	65.6	39.1	3.2	57.7	29.6	4.9	65.5	27.4	5.3	67.3	37.4	3.5	59.2
sedf-2.8	37.9	3.4	58.7	31.5	4.7	63.8	34.9	3.6	61.6	38.1	3.2	28.7	34.4	4	61.5
sedf-250	32.4	5	62.5	35.2	4.5	60.3	34.1	3.5	62.4	39.9	3.1	57	34.3	4.1	61.6
sedf-280	28.5	6.1	65.4	42.1	3.1	54.8	41.3	2.7	56	36.2	3.8	60	34.1	4.6	61.3

Table 5.10: Web Application Domain0 CPU Time. An summary of the percentage of total runtime that Domain0 spent in the blocked, running, and waiting states.

	25 Users			50 Users			100 Users			500 Users			800 Users		
	block	run	wait	block	run	wait	block	run	wait	block	run	wait	block	run	wait
credit-256	21.3	33.6	45.1	37.3	22.4	40.3	28.1	28.3	43.6	36.7	21.7	41.5	30.2	26.7	43.1
credit-512	27.8	28.9	43.3	28	28.7	43.2	31.8	25.6	42.6	37.6	21.2	41.2	33.3	25.5	41.3
credit-1024	28	29.2	42.8	34	24.4	51.1	36.9	22.5	40.6	36.7	22.5	40.6	38.2	20.9	40.8
credit-2048	30.8	27	42.2	28.7	27.9	43.4	36.4	22.3	41.3	31.6	25.4	43	32.8	24.6	42.6
credit-4096	27	50.8	22.2	32.4	25.4	42.2	33.6	24.7	41.7	35.9	22.4	41.8	38.5	20.5	41
sedf-20	36.4	26.3	37.3	42.3	19	38.7	48.8	22.6	28.5	56.1	17.5	26.4	53.2	17.6	29.2
sedf-2.5	40.8	29.3	29.8	43.1	18.9	38.1	49.8	20.4	29.8	43.1	20.9	36	36.7	19.9	43.5
sedf-2.8	45.3	18.9	35.8	52.8	19.7	27.5	50.3	17.1	32.6	45.2	18.3	36.6	42.7	23.2	34.1
sedf-250	49.1	22.5	28.4	47.2	21.9	30.9	45.9	20.1	34	42.2	18.6	39.2	49.5	19.3	31.2
sedf-280	42.1	25.5	32.4	47.4	18.3	34.3	48.1	16.8	35.1	51.3	18.9	29.9	41.1	22.1	36.8

5.3 VPS Results

Due to the time constraints of the the research, it was not possible to complete the VPS application case study tests. The VM image was created as a part of the work and the web application was developed to run on each VPS instance. The results of the study would provide insight into whether the performance of each VPS VM instance could be improved through the adjustment of the scheduler configuration. This remains a point for future research.

5.4 Windows Domain Results

This section presents the results that were obtained from running the Windows Domain tests.

5.4.1 Exchange Results

The Exchange LoadGen tool produced results that were not of a high enough quality for analysis. The tool is aimed at Systems Administrators who want to test the capacity of an Exchange mail server before deploying it in a production environment. The output produced by the tool was neither verbose nor indicative of performance. Given the nature of the results obtained it was not possible to provide an analysis of how CPU scheduler configuration impacted performance.

5.4.2 MOSS Results

The MOSS experiments showed varying levels of throughput and response time. The results showed that for more than 25 users the sEDF scheduler, under various configurations, was best able to produce the highest levels of throughput. This trend can be seen in in Figure 5.3 and Table 5.11. However, as shown in Section 5.2 for the web application results, there was no one particular scheduler configuration that performed best across all of the user loads. The performance of each scheduling algorithm was largely determined by the user load as shown in Figures 5.3 and 5.4.

The relationship between response time and throughput that occurred for the web application held true for the MOSS application. There were some exceptions though, such as with 25 users where the credit-2048 configuration has the highest throughput at 273 requests per second with a response that was also high at 162 ms (as seen in Tables 5.11 and 5.14). This trend can be attributed to that fact that the tasks performed in each test suite was not homogeneous because they were created as part of a larger test suite of requests. There were some task that inherently had longer response times than others and the throughput measurement did not differentiate between the type of task completed. If more of the low response time tasks were completed, then throughput can be higher, even when some other tasks have a higher response time.

The `xenmon` results indicated that the MOSS VM was predominantly CPU bound for most of the scheduler configurations across user loads. In Table 5.17 this trend can be seen where, in general, most of the VM's execution time is spent in a ready state waiting for access to the CPU. This type of behavior can be expected with a web server where little I/O is needed, especially if frequently accessed files are cached in memory. The MSSQL VM that provides storage for the MOSS VM was both CPU and I/O bound across user loads, as seen in Table 5.18 by the large amount of time that was spend blocked and waiting in most scheduler configurations. This is dues to the fact that while databases are stored in memory, frequent access to the disk must still be performed to save the state of the database to persistent memory. An interesting trend was seen with some scheduler configurations where the configuration of the scheduler significantly effected the amount of time a VM blocked. For example, in Table 5.18 with a user load of 25 users the sedf-2 and sedf-2.8 configurations both caused the MSSQL VM to block and wait approximately 50% of the time while the sedf-200 and sedf-175 configurations caused the VM to only block 11% of the time and wait 77% of the time.

An overall trend evident in Table 5.16 is that response time for up to 100 users was a fairly consistent across scheduler configurations. This can be seen in Figure 5.4 and is emphasised by the overlapping confidence intervals. The increase in user load from 100 to 500 users drastically altered this trend, which would seem to indicate a bottleneck in the system. Such a bottleneck could exist in two places, the test client and/or the VMs. While a user load of 500 users on the MOSS server was high, even higher user loads were sustained in the web application tests with much greater constancy maintained with both throughput and response time (as seen in Figures 5.1 and 5.2). This would indicate that the VMs were more the source of the bottleneck, however the client cannot be ruled out entirely. JMeter is a Java application and creates a new thread for each client that it simulates, so with 500 clients being run there are over 500 threads running. However, the task manager was run on the test machine during the test executions and both CPU and memory utilization did not exceed 50%, suggesting that there was indeed excess capacity on the test machine while running high user loads. Unfortunately there is no way to detect if there was thrashing that occurred within the JVM (Java Virtual Machine) as threads competed for access to CPU and memory resources. The `xenmon` results provide some further insight. A user load of 500 users cause each VM and Domain0 to spend an increased amount of time blocked in many of the scheduler configurations as shown in Table 5.17.

The MOSS results point out that the default configuration (credit-256) of credit CPU scheduler led to suboptimal levels of throughput with user loads of more than 25 users. This trend is clearly shown in Figure 5.3. The use of the optimal CPU scheduler configuration resulted in an up to 14% increase in the level of throughput. The largest increase in throughput over the default CPU scheduler configuration was seen with a user load of 100 users as shown in Table 5.11 with a throughput of 265.75 requests per second for the credit-256 configuration and 304.76 requests per second for the sedf-200 configuration.

Table 5.11: MOSS Throughput Results. An overview of the MOSS throughput results.

	25 Users	50 Users	100 Users	500 Users
credit-256	267.24	272.35	265.75	138.61
credit-512	262.55	267.82	268.62	7.31
credit-1024	261.54	265.83	263.05	150.23
credit-2048	**273.08**	264.28	267.00	11.72
sedf-20	206.75	212.40	288.17	163.14
sedf-2	183.63	244.11	257.79	**186.90**
sedf-2.8	188.35	255.35	274.19	69.45
sedf-200	199.46	288.81	**304.76**	65.69
sedf-175	202.88	**291.56**	278.56	159.91

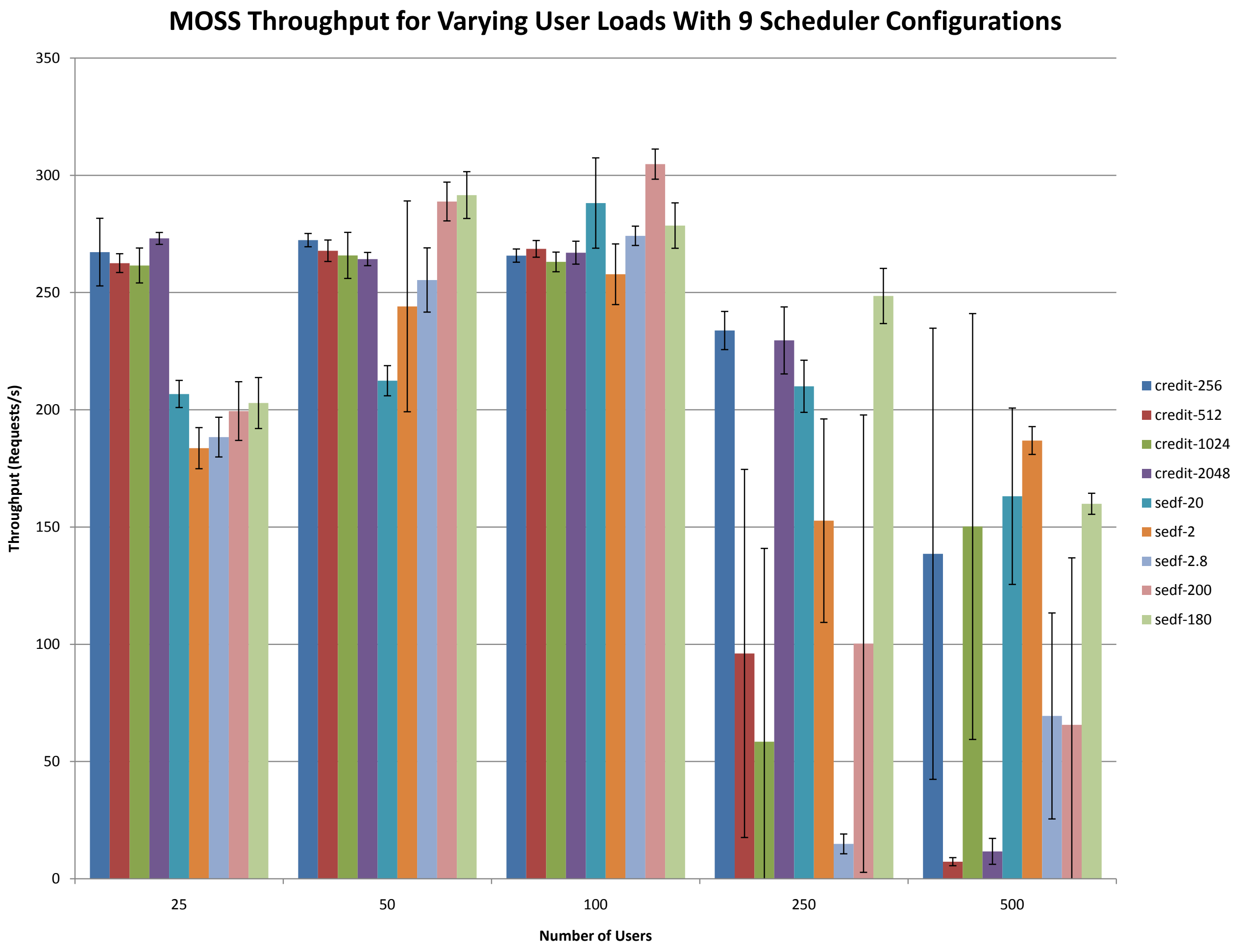


Figure 5.3: MOSS Throughput Graph. A graphical depiction of the average throughput for each scheduler configuration under each user load with 95% confidence intervals.

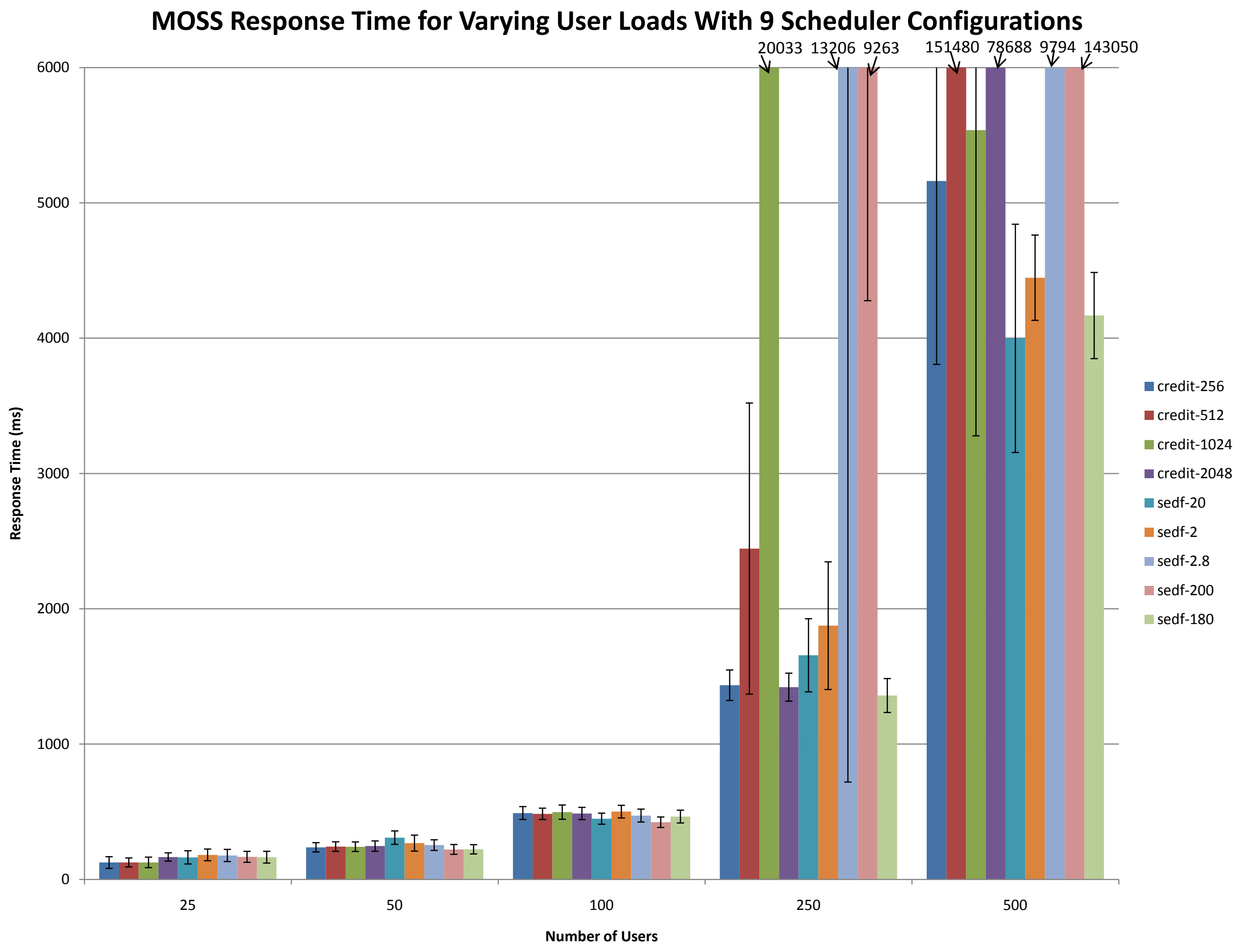


Figure 5.4: MOSS Response Time Graph. A graphical depiction of the average response time for each scheduler configuration under each user load with 95% confidence intervals.

Table 5.12: MOSS Throughput Standard Deviation. An overview of the standard deviation of the throughput values obtained in the MOSS tests.

	25 Users	50 Users	100 Users	500 Users
credit-256	16.45	3.23	3.22	109.74
credit-512	4.57	5.22	4.06	1.96
credit-1024	8.50	11.21	4.78	103.59
credit-2048	2.89	3.22	5.58	6.30
sedf-20	6.60	7.34	21.97	42.90
sedf-2	10.00	51.28	14.75	6.77
sedf-2.8	9.66	15.63	4.70	50.10
sedf-200	14.28	9.43	7.33	81.20
sedf-175	12.40	11.39	11.04	5.12

Table 5.13: MOSS Throughput 95% Confidence Intervals. An overview of the 95% confidence interval of the results obtained in the MOSS tests. Each value indicated in the table is a ± value from the mean.

	25 Users	50 Users	100 Users	500 Users
credit-256	14.42	2.83	2.82	96.19
credit-512	4.01	4.57	3.56	1.72
credit-1024	7.45	9.82	4.19	90.80
credit-2048	2.53	2.82	4.89	5.52
sedf-20	5.78	6.43	19.26	37.61
sedf-2	8.77	44.95	12.93	5.93
sedf-2.8	8.46	13.70	4.12	43.92
sedf-200	12.52	8.27	6.43	71.17
sedf-175	10.87	9.98	9.68	4.49

Table 5.14: MOSS Response Time Results. An overview of the MOSS response time results.

	25 Users	50 Users	100 Users	500 Users
credit-256	**124.63**	237.20	490.59	5161.83
credit-512	**125.32**	242.73	484.54	151480.25
credit-1024	**125.78**	241.73	497.18	5538.13
credit-2048	165.80	246.22	487.16	78688.39
sedf-20	162.97	308.70	448.66	3998.81
sedf-2	181.23	268.19	500.92	4446.99
sedf-2.8	176.94	253.50	471.92	9793.52
sedf-200	166.80	**221.75**	**422.57**	14305.50
sedf-175	164.32	**222.38**	464.43	**4167.24**

Table 5.15: MOSS Response Time Standard Deviation. An overview of the standard deviation of the response times obtained in the MOSS tests.

	25 Users	50 Users	100 Users	500 Users
credit-256	806.08	643.30	883.26	18208.11
credit-512	615.95	656.26	783.92	167301.60
credit-1024	716.36	657.97	975.98	31602.61
credit-2048	575.20	709.18	841.36	114448.48
sedf-20	800.48	824.25	791.10	12286.09
sedf-2	667.94	1052.86	854.54	4916.91
sedf-2.8	703.36	715.97	900.74	17262.55
sedf-200	654.22	703.34	773.30	58447.32
sedf-175	706.18	665.89	900.56	4591.08

Table 5.16: MOSS Response Time 95% Confidence Intervals. An overview of the 95% confidence interval of the response times obtained in the MOSS tests. Each value indicated in the table is a ± value from the mean.

	25 Users	50 Users	100 Users	500 Users
credit-256	43.22	34.18	47.50	1355.65
credit-512	33.33	35.15	41.93	54650.85
credit-1024	38.84	35.37	52.75	2260.22
credit-2048	30.51	38.24	45.15	29453.99
sedf-20	48.81	49.60	40.86	843.50
sedf-2	43.21	59.08	46.67	315.33
sedf-2.8	44.94	39.28	47.70	1816.30
sedf-200	40.61	36.28	38.84	6325.22
sedf-175	43.47	34.19	47.31	318.34

Table 5.17: MOSS VM CPU Time. An summary of the percentage of total runtime that the MOSS VM spent in the blocked, running, and waiting states.

	25 Users			50 Users			100 Users			500 Users		
	block	run	wait	block	run	wait	block	run	wait	block	run	wait
credit-256	8.8	32.4	58.8	2.6	36.7	60.7	2	37.5	60.5	8.1	33.4	58.5
credit-512	3.3	35.5	61.3	2.7	36	61.3	2	36.8	61.2	41.1	8.3	50.5
credit-1024	3.4	34.6	62	1.9	35.9	62.2	1.7	36.4	62	41.2	7.5	51.3
credit-2048	3.6	35.4	61	3.3	35.7	61.1	1.7	37.2	61.2	37.8	11	51.2
sedf-20	4.1	27.3	68.6	49.4	0.5	50.1	1.7	29.6	68.7	12.6	23.4	64
sedf-2	49.3	0.6	50.1	48.7	0.6	50.7	47.9	0.6	51.5	48.2	0.6	51.2
sedf-2.8	3.5	26.4	70.1	49.2	0.6	50.3	2.9	27.7	69.4	8	25.2	66.8
sedf-200	4.9	26.6	68.6	3.7	27.8	68.5	1.8	30	68.2	39.6	7	53.4
sedf-175	5.7	26.7	67.6	2.4	29.5	68	48.1	0.6	51.3	49.3	0.6	50.1

Table 5.18: MOSS MSSQL VM CPU Time. An summary of the percentage of total runtime that the MSSQL VM spent in the blocked, running, and waiting states.

	25 Users			50 Users			100 Users			500 Users		
	block	run	wait	block	run	wait	block	run	wait	block	run	wait
credit-256	33.2	9.6	57.2	30	11	59	30.2	10.9	58.9	33.4	9.7	56.9
credit-512	30.4	10.5	59	30.3	10.6	59.14	30.4	10.8	58.8	55.9	1.1	43
credit-1024	30.8	10.3	58.9	30.3	10.6	59.1	30.4	10.6	59	56.8	0.9	42.3
credit-2048	30.5	10.4	59	30.8	10.3	58.9	30.4	10.6	59	56.8	1.5	41.6
sedf-20	18.4	10.1	71.5	49.6	0.3	50.1	27	10.4	62.6	28.6	8.9	62.3
sedf-2	15	10.5	74.4	20.4	9.1	70.5	20.3	10.4	69.3	49.2	0.3	50.5
sedf-2.8	49.5	0.3	50.2	48.8	0.3	50.9	48.5	0.3	51.1	49	0.3	50.7
sedf-200	10.9	11.3	77.3	16.9	11.8	71.2	25.5	10.9	63.6	44.6	1.3	54.1
sedf-175	11.8	11.7	76.5	26.3	10.6	63.1	49.1	0.3	50.6	49.6	0.3	50.1

Table 5.19: MOSS Domain0 CPU Time. An summary of the percentage of total runtime that Domain0 spent in the blocked, running, and waiting states.

	25 Users			50 Users			100 Users			500 Users		
	block	run	wait	block	run	wait	block	run	wait	block	run	wait
credit-256	45.9	15.7	38.5	42.1	17.3	40.6	42.4	17.1	40.5	46.2	15.3	38.5
credit-512	34.3	17.8	48	34.4	18.1	47.5	34.4	17.9	47.7	65.4	12.6	22.1
credit-1024	31.6	18.2	50.2	30.2	18.5	51.3	30.5	18.4	51	66	5.1	28.9
credit-2048	29.9	38.1	32	29.8	17.6	52.5	28.7	18	53.4	62.5	5.9	31.6
sedf-20	56.9	10.8	32.3	94.8	2.7	2.6	48.6	11.6	39.9	58.1	9.4	32.5
sedf-2	94.3	2.5	3.2	58.4	10.4	31.1	95.3	2.6	2.1	95.3	2.6	2.1
sedf-2.8	94.3	2.5	3.2	94.8	2.6	2.6	49.1	10.7	40.1	95.1	2.6	2.3
sedf-200	62.2	9.9	27.8	56.5	10.7	32.8	47.3	11.7	41	84.9	4.9	10.2
sedf-175	62.2	9.9	28.87	48.2	11.7	40.2	50.6	11.4	38.1	58	9.8	32.2

5.5 A Review of Claims and Contributions

This work successfully completed the intended goals. Its contributions include:

- The creation of three synthetic workloads that can be used in future research. Each synthetic workload was constructed to represent a typical real world workload for Windows and Linux virtual machine environments.
- The measurement of the performance of two scheduling algorithms in various configurations for each case-study workload, using throughput and response time as metrics.
- Determination of which scheduling algorithms and which specific configurations were best suited for each of the synthetic workloads given various target user loads.

A listing of all of the source code used along with the Linux VMs can be released for use in future work. In addition, all of the PowerShell scripts that were used to run the experiments can also be released. This establishes a common open-source framework that other performance studies can use. While the PowerShell scripts have code that is specific to the Xen hypervisor, they can easily be modified to run on other hypervisors. Thus, the workloads can provide a common framework for benchmarking hypervisors.

Aside from the three workloads that were created, the results of the performance studies that were conducted provide insight into the relationship between application performance and the hypervisor CPU scheduler configuration. It suggests that a simple "out of the box" hypervisor, while convenient, is most certainly not optimal.

5.6 Threats to Validity

The steps taken to conduct this research were carefully thought out and planned. The statistical techniques that were employed established confidence in the data. Points of data that have a low level of confidence are clearly stated, as indicated by a large confidence interval. The only values that can potentially be scrutinized are the blocked, wait, and execution time values obtained for each virtual machine. The percentage values were calculated by taking the average of the 5 tests for each scheduler configuration with each user load. Since these values were calculated as percentages of the total execution time and given the high level of indeterminacy of CPU scheduling, it did not seem necessary to calculate standard deviations and confidence intervals. This decision is, however, supported by the fact that in most cases each of the average percentages calculated sum to 100%.

The main threat to validity comes in the form of hardware decisions, which were largely limited by the amount of research funds that were available. The system that was built for this research was a high-end work station computer. However, it was not a server class machine. This distinction should not take away from the results. Even with a more powerful system the hypervisor must still schedule VM access to limited resources such as disk, network, CPU, and memory that all have the caveat of only supporting a limited (if any) amount of parallel access.

5.7 Conclusion

This work highlighted the relationship between application performance and the hypervisor CPU scheduling algorithm that is used. Through an empirical study, it was determined that CPU scheduling is both workload and user load sensitive. Further, simply setting a higher scheduling priority on a virtual machine does not provide any guarantee of increased performance and in fact can lead to a degradation in performance. To increase application performance an evaluation of the bottlenecks in the system must be conducted. Once bottlenecks are identified, a performance study can be conducted to determine methodologies for increasing performance that are based on an understanding of the bottlenecks that exist in a system.

A significant result of this work is the justification for the use of sEDF CPU scheduler in certain situations, as indicated by its ability to provide a consistently better level of performance with the MOSS application for greater than 25 users and a generally better level of performance with the web application. The Xen documentation indicates that the sEDF scheduler should not be used because there are plans to remove it from the standard Xen distribution, a fate which the borrowed virtual time CPU scheduler has already faced. Through the course of this research the case has been made for not only the use of sEDF, but the persistence of the option to use it as part of the standard distribution.

To further demonstrate the significance of the results obtained, suggestions for future work are provided in the next chapter. Specific areas where this work can be expanded and augmented are highlighted as well as how the lessons learned through the performance study can be applied.

Chapter 6

Future Work

"Tell me and I forget. Teach me and I remember. Involve me and I learn."

—Benjamin Franklin

This research produced a set of three workloads that can be used to evaluate the performance of a virtual environment. In the case of this study, the workloads were used to evaluate the performance of various CPU scheduler configurations of the Xen hypervisor. Given that this framework now exists it can be expanded and implemented on other hypervisors. This opens the door to a nearly endless amount of follow-up work.

Within the Xen hypervisor, there are many more combinations of CPU scheduling configurations that can be evaluated. This work only looked at a very small subset. However, the results obtained can provide a baseline from which inferences can be made about other scheduling configurations that have the potential to increase throughput and decrease response time. A similar performance study can then be conducted to see if it is possible to obtain better levels of performance than those found in this study.

An open question that arose from this work was the cause of the variability of the "best" scheduling algorithm configuration based on user load. Further work is required to determine the root cause of this phenomenon as none of the data collected points to a clear cause of the trend. The insertion of some type of OS monitoring tool into each VM could possibly provide the feedback needed to address this question.

Another open question is the effect of the underlying storage system upon VM performance. The count of blocked time, highlighted in the previous chapter, does give a sense of the amount of time that was related to disk I/O. The `xenmon` tool was supposed to output I/O counts, but for whatever reason it did not. It may turn out that greater throughput of the storage system has an effect on the performance of the CPU scheduling algorithms used.

Due to timing restraints of this work only one client machine was able to be utilized for the simulation of users of each workload. However, each of the tools used supports execution in a distributed mode. An extension of this work implemented using multiple test clients is yet another area of future research.

Automatic tuning of the Xen hypervisor CPU scheduler is yet another area of future work. If the results obtained in this study were expanded and augmented they could be used to construct a tuning algorithm that would be capable of reconfiguring the CPU scheduler to maximize performance. Such a system would prove valuable and ensure high utilization of the CPU.

A final point of research that could be conducted would be a verification of the methodologies used in this study. This is not to suggest that these findings are invalid, rather to say that a confirmation of the results would further strengthen the claims that were made. The notion of repeated research is not uncommon and was performed in the case of Xen [3].

This chapter has addressed several key areas of future work based the results obtained in this study. As hardware continues to decrease in price while increasing in speed, there will surely be an even larger move to consolidate physical machines through the use of virtualization. The continued success of virtualization will rely on research like this that asks tough questions about performance and provides a carefully constructed empirical study to determine the answer.

Appendix A

Xen Commands

This appendix is meant to serve as a guide to using Xen. Various tasks that were necessary to conduct the building of the Xen environment are outlined as well as several management tasks.

A.1 Xen Installation

Ubuntu was used as the base operating system primarily for the availability of pre-compiled packages. Thus, the installation was facilitated by the aptitude package manager. The package manager automatically resolves all of the dependencies of the packages that it installs. The following command was used to install the base Xen Server package and the optional GUI interface:

```
sudo apt-get install ubuntu-xen-server virt-manager
```

The command downloads all of the required Xen files and dependencies. A new kernel image is created that is set to load the Xen kernel module to facilitate the running of virtual machines. The `virt-manger` program was also installed to allow for a GUI to create and run virtual machines.

A.2 Creating a VM

To create virtual machines for testing the `virt-manger` tool was used. The GUI provides a set of configuration options and then creates a VM. There are additional tools included with the `xm` command that facilitate manually creating a VM.

A.3 Cloning a VM

Several of the base OS installs were greatly aided by the clone command. A template of a base operating system with updates installed and configured was cloned to create new VMs. The ability to clone VMs dramatically reduced the time required to create each workload and ensured a standard configuration across VMs in each workload. Clone functionality is provided via the `xm` command. The syntax of the command is demonstrated below.

```
xm clone source_vm detination_location
```

A.4 Increase the Number of Loopback Devices

Xen VMs can be setup to use flat files as disk images. When this disk configuration is used, the OS creates a loopback device to mount the disk image to the file system. By default the Ubuntu Xen install only has 4 loopback devices enabled, which restricts the number of VMs that can be run to 4. A simple edition of the following to the /etc/modules file (if the file does not exist, it will need to be created) will increase the number of loopback devices to *xx*.

```
add loop max_loop=xx
```

The number of available loopback devices can be seen by using the *ls* command.

```
ls -l /dev/loop*
```

A.5 Set the Xen CPU Scheduler Type

The Xen scheduler can be changed by setting the `sched` kernel boot parameter. By default the credit scheduler is used for CPU scheduling, but setting `sched=sedf` will specify that the sEDF CPU scheduler should be used. The easiest way to implement this parameter is to copy the existing Xen boot menu entry in the `menu.lst` file, which is typically located in the `/boot/grub` folder. The file will need to be edited by the root user account. An example of the modified boot entry is provided below.

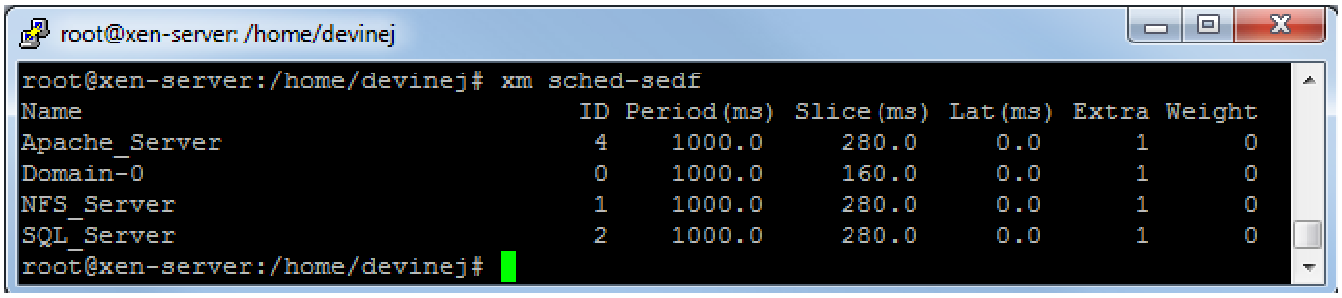
```
title     Xen 3.3 / Ubuntu 8.04.3 LTS, kernel 2.6.24-26-xen sEDF
root      (hd1,0)
kernel    /boot/xen-3.3.gz sched=sedf
module    /boot/vmlinuz-2.6.24-26-xen root=UUID=e8c13945-5158
module    /boot/initrd.img-2.6.24-26-xen
quiet
```

A.6 Determining the Scheduler Configuration

Xen has 2 commands built into the `xm` command that allows the scheduler parameters to be seen. For the credit scheduler the command is `xm sched-credit` and for the sedf scheduler the command is `xm sched-sedf` (sample output can be seen in Figure A.1.

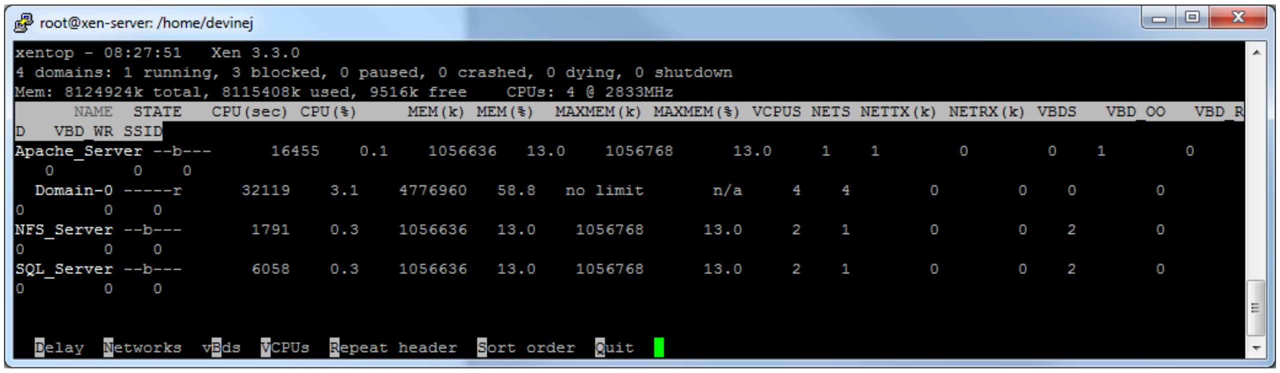
A.7 Xentop

Xen has a built in top command called `xentop`. The command gives CPU and memory utilization like the typical `top` command, but with a listing of VMs. An example of the command is given in FigureA.2.



```
root@xen-server:/home/devinej# xm sched-sedf
Name                                ID  Period(ms)  Slice(ms)  Lat(ms)  Extra  Weight
Apache_Server                        4      1000.0      280.0      0.0      1       0
Domain-0                             0      1000.0      160.0      0.0      1       0
NFS_Server                           1      1000.0      280.0      0.0      1       0
SQL_Server                           2      1000.0      280.0      0.0      1       0
root@xen-server:/home/devinej#
```

Figure A.1: `sched-credit` Command. An example of the `xm sched-credit` command output.



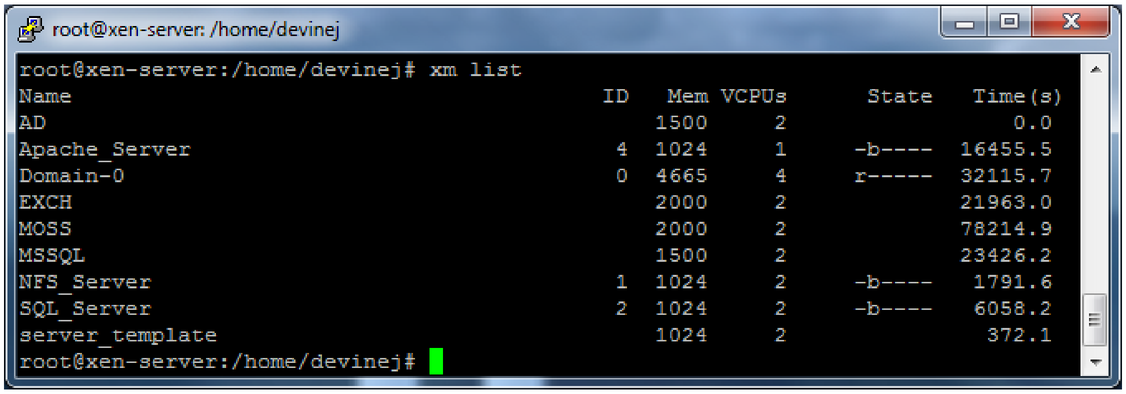
```
xentop - 08:27:51   Xen 3.3.0
4 domains: 1 running, 3 blocked, 0 paused, 0 crashed, 0 dying, 0 shutdown
Mem: 8124924k total, 8115408k used, 9516k free    CPUs: 4 @ 2833MHz
      NAME  STATE   CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS NETS NETTX(k) NETRX(k) VBDS   VBD_OO   VBD_R
D   VBD_WR SSID
Apache_Server --b---      16455    0.1    1056636   13.0    1056768      13.0     1    1        0        0    1        0
   0        0    0
  Domain-0 -----r      32119    3.1    4776960   58.8   no limit       n/a     4    4        0        0    0        0
0        0    0
NFS_Server --b---       1791    0.3    1056636   13.0    1056768      13.0     2    1        0        0    2        0
0        0    0
SQL_Server --b---       6058    0.3    1056636   13.0    1056768      13.0     2    1        0        0    2        0
0        0    0


  Delay  Networks  vBds  VCPUs  Repeat header  Sort order  Quit
```

Figure A.2: `xentop` Command. An example of the `xentop` command output.

A.8 Viewing the Running Virtual Machines

A list of running virtual machines can be viewed using the `xm list` command. A sample of the output is provided in Figure A.3.



```
root@xen-server:/home/devinej# xm list
Name                                        ID   Mem VCPUs      State   Time(s)
AD                                               1500     2                0.0
Apache_Server                                4   1024     1     -b----  16455.5
Domain-0                                     0   4665     4     r-----  32115.7
EXCH                                             2000     2             21963.0
MOSS                                             2000     2             78214.9
MSSQL                                            1500     2             23426.2
NFS_Server                                   1   1024     2     -b----   1791.6
SQL_Server                                   2   1024     2     -b----   6058.2
server_template                                  1024     2               372.1
root@xen-server:/home/devinej# 
```

Figure A.3: `xm list` Command. An example of the `xm list` command output.

Appendix B

Tools

This appendix was created to provide a description of all the tools that were used in this research and summarize their functionality. Where appropriate, sample commands of each tool are given.

B.1 JMeter

JMeter is an a performance benchmarking tool for websites that is part of the Apache project. The tool is written in Java. It allows for simultaneous connections to a webserver to be initialized by creating a new thread for each connection. JMeter was the tool of choice for benchmarking MOSS due to its ability to run a proxy server to record web browser events. Once the events were recorded through the proxy server, they were added to a test suite.

B.2 LoadGen

`LoadGen` is a Microsoft Exchange performance testing tool. It was created to allow system administrators to performance test new Exchange installations to determine if they can handle the anticipated load of a new environment that is being deployed.

LoadGen initiates a series of client connections to the Exchange server and performs common tasks, such as sending emails, logging in and out, creating calendar events, etc.

B.3 Pageant

Pageant is an open source tool that allows for the creating of public authentication keys for putty and plink ssh sessions [13]. This tool allowed a key to be created so that the root password did not need to be entered in clear text when scripting the experimentation. Once a share key is created with the program, it is added to the `authorized_keys` file on the server. Once the key is added a password is not needed to `ssh` into the server.

B.4 Plink

Plink is an open source command line interface to PuTTY, a Windows Telnet and SSH client [13]. Plink was used to execute commands on the Xen server from the Windows test machine. The tool can be accessed at the project webpage.

B.5 Ubuntu

Ubuntu is an open-source Linux distribution based on Debian Linux. It features a robust package manager system, called aptitude, that allows for the easy installation of new software by automatically resolving dependencies. For this reason, Ubuntu was chosen for this research. Ubuntu 9.04 Server Edition was the primary OS on the Xen server and each of the web application and VPS application workloads.

B.6 virt-manager

`Virt-manager` is an open source monitoring and management GUI for virtual machines. It was used to aid in the installation and maintenance of the virtual machines used in the research.

B.7 Xenmon

`Xenmon` is a performance monitoring tool for Xen. `Xenmon` reports I/O and CPU scheduling information. It can run interactively or log data directly to a file. `Xenmon` was crucial in measuring the effectiveness of each CPU scheduler and configuration used during the research. The tool is part of the Xen installation. It can be invoked using the following command:

```
xenmon.py -p filename_prefix -i logging_interval
          -t time_to_log_for -n
```

The example above specifies a prefix to the output file name, the interval at which information is written to the log file (i), the total time that `xenmon` is run for(t), and that no information should be output to the terminal (n). A separate output file is created for each domain.

B.8 Xen

Xen is an open source hypervisor. Xen was used because to has the ability to use different CPU scheduling algorithms and allows the user to configure parameters for each scheduling algorithm.

Appendix C

Source Code

C.1 Test Configuration Files

C.1.1 Web Application JMeter Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="2.1">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Test
        " enabled="true">
      <stringProp name="TestPlan.comments"></stringProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
      <elementProp name="TestPlan.user_defined_variables" elementType="
          Arguments" guiclass="ArgumentsPanel" testclass="Arguments"
          testname="User Defined Variables" enabled="true">
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="TestPlan.user_define_classpath"></stringProp>
    </TestPlan>
    <hashTree>
      <ThreadGroup guiclass=”ThreadGroupGui” testclass=”ThreadGroup” testname=”Thread Group” enabled=”true”>
        <elementProp name=”ThreadGroup.main_controller” elementType=”LoopController” guiclass=”LoopControlPanel” testclass=”LoopController” testname=”Loop Controller” enabled=”true”>
          <boolProp name=”LoopController.continue_forever”>false</boolProp>
          <intProp name=”LoopController.loops”>-1</intProp>
        </elementProp>
        <stringProp name=”ThreadGroup.num_threads”>25</stringProp>
        <stringProp name=”ThreadGroup.ramp_time”>1</stringProp>
        <longProp name=”ThreadGroup.start_time”>1269900472000</longProp>
        <longProp name=”ThreadGroup.end_time”>1269900472000</longProp>
        <boolProp name=”ThreadGroup.scheduler”>true</boolProp>
        <stringProp name=”ThreadGroup.on_sample_error”>continue</stringProp>
        <stringProp name=”ThreadGroup.duration”>1800</stringProp>
        <stringProp name=”ThreadGroup.delay”></stringProp>
      </ThreadGroup>
      <hashTree>
        <ConfigTestElement guiclass=”HttpDefaultsGui” testclass=”ConfigTestElement” testname=”HTTP Request Defaults” enabled=”true”>
          <elementProp name=”HTTPsampler.Arguments” elementType=”Arguments” guiclass=”HTTPArgumentsPanel” testclass=”Arguments” testname=”User Defined Variables” enabled=”true”>
            <collectionProp name=”Arguments.arguments”/>
          </elementProp>
          <stringProp name=”HTTPSampler.domain”></stringProp>
          <stringProp name=”HTTPSampler.port”>80</stringProp>
          <stringProp name=”HTTPSampler.connect_timeout”></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path"></stringProp>
        </ConfigTestElement>
        <hashTree/>
        <HTTPSampler guiclass="HttpTestSampleGui" testclass="HTTPSampler
        " testname="HTTP Request" enabled="true">
          <elementProp name="HTTPsampler.Arguments" elementType="
              Arguments" guiclass="HTTPArgumentsPanel" testclass="
              Arguments" enabled="true">
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">10.0.0.20</stringProp>
          <stringProp name="HTTPSampler.port"></stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path">/index.php</stringProp>
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.follow_redirects">false</boolProp>
          <boolProp name="HTTPSampler.auto_redirects">true</boolProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
          <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp
              >
          <stringProp name="HTTPSampler.FILE_NAME"></stringProp>
          <stringProp name="HTTPSampler.FILE_FIELD"></stringProp>
          <stringProp name="HTTPSampler.mimetype"></stringProp>
          <boolProp name="HTTPSampler.monitor">false</boolProp>
          <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
        </HTTPSampler>
```

```
      <hashTree/>
      <ResultCollector guiclass=”GraphAccumVisualizer” testclass=”ResultCollector” testname=”Graph Full Results” enabled=”true”>
        <boolProp name=”ResultCollector.error_logging”>false</boolProp>
        <objProp>
          <name>saveConfig</name>
          <value class=”SampleSaveConfiguration”>
            <time>true</time>
            <latency>true</latency>
            <timestamp>true</timestamp>
            <success>true</success>
            <label>true</label>
            <code>true</code>
            <message>true</message>
            <threadName>true</threadName>
            <dataType>true</dataType>
            <encoding>false</encoding>
            <assertions>true</assertions>
            <subresults>true</subresults>
            <responseData>false</responseData>
            <samplerData>false</samplerData>
            <xml>true</xml>
            <fieldNames>false</fieldNames>
            <responseHeaders>false</responseHeaders>
            <requestHeaders>false</requestHeaders>
            <responseDataOnError>false</responseDataOnError>
            <saveAssertionResultsFailureMessage>false</saveAssertionResultsFailureMessage>
            <assertionsResultsToSave>0</assertionsResultsToSave>
            <bytes>true</bytes>
```

```
              </value>
            </objProp>
            <stringProp name="filename">C:\Documents and Settings\
                Administrator.DOMAIN\Desktop\web_app_scripts\test_results.
                jtl</stringProp>
            <boolProp name="ResultCollector.success_only_logging">true</
                boolProp>
          </ResultCollector>
          <hashTree/>
        </hashTree>
      </hashTree>
    </hashTree>
</jmeterTestPlan>
```

C.2 Test Scripts

C.2.1 Web Application sEDF Tests Powershell Script

```
# James Devine
#
# web_test_sedf.ps1 - powershell script to run the web application
    workload with various
#                       configurations of the sedf scheduler
#

$users=@(25, 50, 100, 500, 800)

#loop through each userload
for ($k=0; $k -le $users.Length - 1; $k++)


{
    $userNum=$users[$k]

    #loop through scheduler configurations
    $dom0_slices=@(15, 2.5, 1.6, 250, 160)
    $dom0_periods=@(20, 10, 10, 1000, 1000)
    $domU_periods=@(100, 10, 10, 1000, 1000)
    $test_slices=@(20, 2.5, 2.8, 250, 280)

    for ($j=0; $j -le $test_slices.length-1; $j++)
    {
        #set the period and slice variables
        $domU_period=$domU_periods[$j]
        $dom0_slice=$dom0_slices[$j]
        $dom0_period= $dom0_periods[$j]
        $test_slice=$test_slices[$j]

        #set the scheduler periods and slices
        plink -l root -i "C:\Documents and Settings\Administrator.DOMAIN
            \Desktop\key.ppk" 10.0.0.1 "xm sched-sedf Apache_Server -p
            $domU_period -s $test_slice"
        plink -l root -i "C:\Documents and Settings\Administrator.DOMAIN
            \Desktop\key.ppk" 10.0.0.1 "xm sched-sedf NFS_Server -p
            $domU_period -s $test_slice"
        plink -l root -i "C:\Documents and Settings\Administrator.DOMAIN
            \Desktop\key.ppk" 10.0.0.1 "xm sched-sedf SQL_Server -p
            $domU_period -s $test_slice"
        plink -l root -i "C:\Documents and Settings\Administrator.DOMAIN
            \Desktop\key.ppk" 10.0.0.1 "xm sched-sedf Domain-0 -p
            $dom0_period -s $dom0_slice"
        plink -l root -i "C:\Documents and Settings\Administrator.DOMAIN
            \Desktop\key.ppk" 10.0.0.1 "xm sched-sedf"

    #loop 5 times for each user load & scheduler config
    $i = 1
    while ($i -le 5)
    {
        #setup the database
        start-job "C:\Documents and Settings\Administrator.DOMAIN\
            Desktop\web_app_scripts\setup.ps1"

        $options="-n -t 1800 -i 1000 -p sedf_domUperiod-$domUperiod-
            TEST-slice-$test_slice-users-$userNum-app-$i"
        #run the experiment
        echo "Running xenmon with the options: $options"
        $RESULT = start-job "C:\Documents and Settings\Administrator
            .DOMAIN\Desktop\web_app_scripts\jmeter-$userNum.ps1"
        #run xenmon
        $out = plink -l root -i "C:\Documents and Settings\
            Administrator.DOMAIN\Desktop\key.ppk" 10.0.0.1 "xenmon.
            py $options"
        #wait for the current job to finish
        wait-job $RESULT

        #move and rename the results file
        mv "C:\Documents and Settings\Administrator.DOMAIN\Desktop\
            web_app_scripts\test_results.jtl" "C:\Documents and
            Settings\Administrator.DOMAIN\Desktop\web_app_sedf\
            sedf_slice-$test_slice-users-$userNum-app-$i.jtl"

        #teardown the database
        start-job "C:\Documents and Settings\Administrator.DOMAIN\
            Desktop\web_app_scripts\teardown.ps1"

        #increment the counter
```

```
            $i++
        }
    }
}
```

C.2.2 Web Application Credit Tests Powershell Script

```


$users=@(50, 100, 500, 800)

#loop through each userload
for ($k=0; $k -le $users.Length - 1; $k++)
{
    $userNum=$users[$k]

    #loop through scheduler configurations
    $weights=@(256, 512, 1024, 2048, 4096)
    #$dom0_slices=@(15, 2, 1.8, 200, 175)
    #$dom0_periods=@(20, 10, 10, 1000, 1000)
    #$domU_periods=@(100, 10, 10, 1000, 1000)
    #$test_slices=@(20, 2, 2.8, 200, 175)
    #$other_slices=@(20, 2, 1.8, 200, 175)

    for ($j=0; $j -le $weights.length-1; $j++)
    {
        $weight=$weights[$j]
        #$domU_period=$domU_periods[$j]
        #$dom0_slice=$dom0_slices[$j]
        #$dom0_period= $dom0_periods[$j]
        #$exch_slice=$exch_slices[$j]
        #$other_slice=$other_slices[$j]

        #set the scheduler weight
        plink -l root -i "C:\Documents and Settings\Administrator.DOMAIN
            \Desktop\key.ppk" 10.0.0.1 "xm sched-credit -d NFS_Server -w
             $weight"
        plink -l root -i "C:\Documents and Settings\Administrator.DOMAIN
            \Desktop\key.ppk" 10.0.0.1 "xm sched-credit -d SQL_Server -w
             $weight"
        plink -l root -i "C:\Documents and Settings\Administrator.DOMAIN
            \Desktop\key.ppk" 10.0.0.1 "xm sched-credit -d Apache_Server
             -w $weight"
        #plink -l root -i "C:\Documents and Settings\Administrator.
            DOMAIN\Desktop\key.ppk" 10.0.0.1 "xm sched-sedf EXCH -p
            $domU_period -s $exch_slice"
        #plink -l root -i "C:\Documents and Settings\Administrator.
            DOMAIN\Desktop\key.ppk" 10.0.0.1 "xm sched-sedf AD -p
            $domU_period -s $other_slice"
        #plink -l root -i "C:\Documents and Settings\Administrator.
            DOMAIN\Desktop\key.ppk" 10.0.0.1 "xm sched-sedf MSSQL -p
            $domU_period -s $test_slice"
        #plink -l root -i "C:\Documents and Settings\Administrator.
            DOMAIN\Desktop\key.ppk" 10.0.0.1 "xm sched-sedf MOSS -p
            $domU_period -s $test_slice"
        #plink -l root -i "C:\Documents and Settings\Administrator.
            DOMAIN\Desktop\key.ppk" 10.0.0.1 "xm sched-sedf Domain-0 -p
            $dom0_period -s $dom0_slice"
        #plink -l root -i "C:\Documents and Settings\Administrator.
            DOMAIN\Desktop\key.ppk" 10.0.0.1 "xm sched-sedf"

        #loop 5 times for each user load & scheduler config
        $i = 1
        while ($i -le 5)
        {
            #setup the database
            start-job "C:\Documents and Settings\Administrator.DOMAIN\
                Desktop\web_app_scripts\setup.ps1"

            $options="-n -t 1800 -i 1000 -p credit_weight-$weight-users-
                $userNum-web_app-$i"
            #$options="-n -t 1800 -i 1000 -p sedf_domUperiod-$domUperiod
                -EXCH-slice-$exch_slice-users-$userNum-exchange-$i"
            #run the experiment
            echo "Running xenmon with the options: $options"
            $RESULT = start-job "C:\Documents and Settings\Administrator
                .DOMAIN\Desktop\web_app_scripts\jmeter-$userNum.ps1"
            #run xenmon
            $out = plink -l root -i "C:\Documents and Settings\
                Administrator.DOMAIN\Desktop\key.ppk" 10.0.0.1 "xenmon.
                py $options"
            #wait for the current job to finish
            wait-job $RESULT

            #move and rename the results file
            mv "C:\Documents and Settings\Administrator.DOMAIN\Desktop\
                web_app_scripts\test_results.jtl" "C:\Documents and
                Settings\Administrator.DOMAIN\Desktop\web_app_credit\
                credit_$weight-users-$userNum-app-$i.jtl"

            #teardown the database
            start-job "C:\Documents and Settings\Administrator.DOMAIN\
                Desktop\web_app_scripts\teardown.ps1"

            #increment the counter
            $i++
            }
        }
}
```

C.3 Web Application Source Code

C.3.1 Main Website

```
<?php
            //include the config file
        include("config.php");

            //start a new session
            session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Web Application Benchmark</title>
<link href="style.css" rel="stylesheet" type="text/css" />
<script type="text/javascript">
<!--
function MM_swapImgRestore() { //v3.0
    var i,x,a=document.MM_sr;
    for(i=0;a&&i<a.length&&(x=a[i])&&x.oSrc;i++) x.src=x.oSrc;
}
function MM_preloadImages() { //v3.0
    var d=document; if(d.images){ if(!d.MM_p) d.MM_p=new Array();
        var i,j=d.MM_p.length,a=MM_preloadImages.arguments;
            for(i=0; i<a.length; i++)
```

```
      if (a[i].indexOf("#")!=0){ d.MM_p[j]=new Image;
          d.MM_p[j++].src=a[i];}}
}

function MM_findObj(n, d) { //v4.01
  var p,i,x;  if(!d) d=document;
  if((p=n.indexOf("?"))>0&&parent.frames.length) {
    d=parent.frames[n.substring(p+1)].document;
          n=n.substring(0,p);}
  if(!(x=d[n])&&d.all) x=d.all[n];
  for (i=0;!x&&i<d.forms.length;i++) x=d.forms[i][n];
  for(i=0;!x&&d.layers&&i<d.layers.length;i++)
  x=MM_findObj(n,d.layers[i].document);
  if(!x && d.getElementById) x=d.getElementById(n); return x;
}

function MM_swapImage() { //v3.0
  var i,j=0,x,a=MM_swapImage.arguments;
  document.MM_sr=new Array;
  for(i=0;i<(a.length-2);i+=3)
   if ((x=MM_findObj(a[i]))!=null)
   {document.MM_sr[j++]=x;
   if(!x.oSrc) x.oSrc=x.src; x.src=a[i+2];}
}
//-->
</script>
</head>

<body class="oneColFixCtr"onLoad=
"MM_preloadImages('image/market_ovr.jpg',
 'image/brand_ovr.jpg','image/home_ovr.jpg',
 'image/promote_ovr.jpg','image/catalogs_ovr.jpg',
'image/about_ovr.jpg','image/blog_ovr.jpg'),
'image/testimonials_ovr.jpg','image/contact_ovr.jpg'">

<div id="container">
<?php include "header.php"?>
<div id="home_body"><br />
<?php
        //simulate a user sign in query//

        //generate a random number
        $rand=rand(0,4);

        if($rand==0)
                $usern="user1@fakehost.com";
        elseif($rand==1)
                $usern="user2@fakehost.com";
        elseif($rand==2)
                $usern="user3@fakehost.com";
    elseif($rand==3)
                $usern="user4@fakehost.com";
        elseif($rand==4)
                $usern="user5@fakehost.com";

        //set the password
        $password1="password";

        //find the user in the database and check password
        //connect to the database
        $conn = mysql_connect($host, $user, $password) or
        die('Error connecting to mysql');
        mysql_select_db($database);

        //build the SQL query
        $sql = "SELECT *".
                "FROM users".
                    "WHERE email = '$usern' AND".
                    "password = MD5('$password1')";

        //execute the SQL query
        $result = mysql_query($sql) or die('Query failed. '
                                                                        .
                                                                          mysql_
                                                                          ()
                                                                          )
                                                                          ;

        //if there is a result the user is in the database
        //and has provided a correct password
        if (mysql_num_rows($result) == 1 )
        {
                // set the session
                $_SESSION['logged_in'] = true;
        }

        //set the username
        $_SESSION['userId']=$usern;

        //log the user's visit//
        //build the SQL query
        $sql = "INSERT INTO log (user) VALUES('$usern')";

        //disconnect from the database
        mysql_close($conn);

?>

<?php
        //code to load content from the database//
        $usern=$_SESSION['userId'];
        print "<b>Welcome to a benchmarking test website $usern</b>";

        //create a database connection
        $conn = mysql_connect($host, $user, $password) or
        die('Error connecting to mysql');
        mysql_select_db($database);

        //build query
        $sql = "SELECT *".
                "FROM content".
                        "WHERE page = 'home'";

        //execute the query and save the result
        $result = mysql_query($sql) or die('Query failed. ' .
                                                                                mysql_err
                                                                                   ()
                                                                                   )
                                                                                   ;

    $row = mysql_fetch_array($result) or die(mysql_error());

        //get trid of slashes used to save the html in the db
        $html=stripslashes($row[1]);

        //close the database connection
        mysql_close($conn);
        print "$html";
?>

</div>
<br /><br />
<!-- end #container -->
<?php include "footer.html"; ?>
<!-- end #container -->
</div>
</body>
</html>
```

C.3.2 SQL Setup Script

```
<?php
        include("config.php");

        //create database connection
        $conn = mysql_connect($host, $user, $password) or die('Error
            connecting to mysql');

        //create the db
        $sql="CREATE DATABASE $database";
        mysql_query($sql) or die('Query failed. ' . mysql_error());

        //select the database
        mysql_select_db($database);

        //create the content table
        $sql="CREATE TABLE content (
                page varchar(20) NOT NULL,
                html longtext NOT NULL,
                PRIMARY KEY  (page)
                ) ENGINE=MyISAM DEFAULT CHARSET=utf8;";
        mysql_query($sql) or die('Query failed. ' . mysql_error());

        //create the users table
        $sql="CREATE TABLE users (
                email varchar(50) NOT NULL,
                password varchar(32) NOT NULL,
                Name mediumtext NOT NULL,
                Address mediumtext NOT NULL,
                Date_Joined timestamp NOT NULL default CURRENT_TIMESTAMP
                    ,
                PRIMARY KEY  (email)
                ) ENGINE=MyISAM DEFAULT CHARSET=utf8;";
        mysql_query($sql) or die('Query failed. ' . mysql_error());

        //create the log table
        $sql="CREATE TABLE log (
                id int NOT NULL auto_increment,
                access_time timestamp NOT NULL default CURRENT_TIMESTAMP
                    ,
                user varchar(50) NOT NULL,
                PRIMARY KEY  (id)
                ) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;"
                    ;
        mysql_query($sql) or die('Query failed. ' . mysql_error());

        //add users
        for($i=0;$i<5;$i++)
        {
                $user="user$i@fakehost.com";
```

```

            //build the query
            $sql="INSERT INTO  users (
                    email ,
                    password ,
                    Name ,
                    Address ,
                    Date_Joined
                    )
                    VALUES (
                    '$user@', MD5('password') ,  'Fake User $i',
                        '123 Fake St #$i
                    Meadville PA, 16335', CURRENT_TIMESTAMP);";
            //execute the query
            mysql_query($sql) or die('Query failed. ' . mysql_error
                ());
    }

    //the html string
    //addslashes() is used so the html formatting does not make
        MySQL angry
    $html=addslashes("<p>This is some html from the db.</p> <p>It's
        really good
            news that this html is visable. It is being loaded from
                the MySQL database.</p>
            <h1>Here is a table</h1><table border='1'><tr><td>
                Column1</td><td>Column2</td></tr>
            <tr><td>Some</td><td>Text</td></tr><tr><td>Some</td><td>
                More text</td></tr>
            <tr><td>The</td><td>Goals</td></tr><tr><td>Is</td><td>To
                </td></tr>
```

```
                    <tr><td>Load</td><td>a respectable</td></tr><tr><td>
                        amount of</td><td>text</td></tr>
                    <tr><td>from</td><td>the</td></tr><tr><td>MySQL</td><td>
                        database</td></tr></table>
                    <h2>This is a heading 2 format</h2>
                    <h3>This is a heading 3 format</h3>
                    <h4>This is a heading 4 format</h4>
                    <h5>This is a heading 5 format</h5>
                    This is the end of the html text that was loaded from
                        the database");

          //add html to the content database
          $sql="INSERT INTO  content (
                    page ,
                    html
                    )
                    VALUES(
                 'home', '$html');";
          //execute the query
          mysql_query($sql) or die('Query failed. ' . mysql_error());

          //close the database connection
          mysql_close($conn);

          echo "The database is all setup!";

?>
```

C.3.3 Powershell Script to Execute the SQL Setup

```
# James Devine's Senior Thesis
#
```

```
# setup.ps1 - A powershell script to launch Internet
#             Explorer to execute the SQL setup script
#


cd "C:\Program Files\Internet Explorer"

#point ie to the setup script
./iexplore.exe 10.0.0.20/setup.php

#sleep for one second
sleep 1

#kill internet explorer
get-process iexplore | stop-process
```

C.3.4 SQL Teardown Script

```
<?php
        include("config.php");

        //connect to the server
        $conn = mysql_connect($host, $user, $password) or die('Error
            connecting to mysql');

        //build the query
        $sql = "DROP database $database;";

        //execute the query
        mysql_query($sql) or die('Query failed. ' . mysql_error());

        echo "database dropped";
```

```
?>
```

C.3.5 Powershell Script to Execute the SQL Teardown

```
# James Devine's Senior Thesis
#
# teardown.ps1 - A powershell script to launch Internet
#                Explorer to execute the SQL teardown script
#


cd "C:\Program Files\Internet Explorer"

#point ie to the teardown script
./iexplore.exe 10.0.0.20/teardown.php

#sleep for one second
sleep 1

#kill internet explorer
get-process iexplore | stop-process
```

C.3.6 SQL Configuration File

```
<?php

        $host="10.0.0.22";
        $database="test";
        $user="root";
        $password="allegheny";

?>
```

C.4 JMeter Log File Reduction

C.4.1 JMeter Log File to CSV Converter

```
"""
Description : Split JTL file into a comma delimited CVS
by          : Oliver Erlewein (c)2008
Date        : 04.02.2008
Lang        : Python 2.4+

MODIFICATIONS:
            James Devine - Modified the functionality of the script on
                                4-1-2010 to return only the time each
                                    request
                                took to process

JMeter JTL field contents:

Attribute & Content
by          Bytes
de          Data encoding
dt          Data type
ec          Error count (0 or 1, unless multiple samples are aggregated)
hn          Hostname where the sample was generated
lb          Label
lt          Latency = time to initial response (milliseconds) - not all
    samplers support this
na          Number of active threads for all thread groups
ng          Number of active threads in this group
rc          Response Code (e.g. 200)
rm          Response Message (e.g. OK)
s           Success flag (true/false)
```

```
sc        Sample count (1, unless multiple samples are aggregated)
t         Elapsed time (milliseconds)
tn        Thread Name
ts        timeStamp (milliseconds since midnight Jan 1, 1970 UTC)
"""

import sys
import re
import datetime
import time

startTime = time.time()
cnt                          = 0
cnt2                         = 0
failCnt                      = 0
reCompile = re.compile("\s([^\s]*?)=\"(.*?)\"")
delimiterCharacterOut = ","

def writeCSVLine(line):
    x = reCompile.findall(line)
    a = dict((row[0], row[1]) for row in x)
    try:
        a['ts'] = str(int(int(a['ts'])/1000))
        x = str(datetime.datetime.fromtimestamp(float(a['ts'])))[0:19]

            #BEGIN MODIFICATIONS
        b =a['t']+ "\n"
            #END MODIFIICATIONS

    except:
        return -1
    o.write(b)
    return 1

print "Splitting JTL file"

try:
    runArgv           = sys.argv            # Save the command line
    jtlInfile         = str(sys.argv[1])    # Name of JTL input file
    cvsOutfile        = str(sys.argv[2])    # Name of CVS output file
    reFilter          = str(sys.argv[3])    # Filter the labels (lb)
        for the filter
except:
    print "Error: Input format: <input file> <output file> <Filter by
        regular expression>"
    raise

try:
    f = open(jtlInfile, "r")
    o = open(cvsOutfile, "w")
except:
    raise

print "Filtering on regular expression : " + reFilter
cmpFilter = re.compile(reFilter)

for line in f:
    try:
        if cmpFilter.search(line):
            returnVal = writeCSVLine(line)
            if returnVal < 0:
                failCnt += 1
            else:
                cnt2 += 1
    except:
        print 'Error in line : ', cnt, line
        raise

    cnt += 1

endTime = time.time()
print "Time taken                    : ", str(endTime-startTime)
print "Lines processed               : ", cnt
print "Lines that passed the filter : ", cnt2
print "Lines skipped (error?)        : ", failCnt

f.close()
o.close()
```

C.4.2 Script to Run the JMeter Converter

```
#!/bin/bash
# James Devine's Senior Thesis
# jtl_to_csv.sh - a bash script that takes to parameters
                              a directory and a delimiter
                              The scripts run program.py with
                              the given delimiter on each
                              file in a directory to produce
                              csv summary files


#get CWD
dir=$(pwd)

#get the directory from a command line arg
directory=$1
```

```
delimiter=$2

cd $directory

#get a list of the files in the directory
FILES="*"

#loop through each file
for f in `ls`
do
          file=$f
          if [ $file!="csv" ]; then
                    python "$dir/program.py" "$file" "$file.csv"
                                                       "$delimiter"
          fi
done
```

C.5 R Code

C.5.1 Xenmon Output Processing

```
# James Devine's Senior Thesis
# cpu_time.R - an R script to calculate the wait, cpu usage, and block
                         time of a domain from xenmon log files

calcPercent=function(location, prefix, domain)
{
          #read each data file into a table

          table1 <- read.table(paste(location,prefix,"-1-dom",domain,".log
               ",sep=""))
```

```
        table2 <- read.table(paste(location,prefix,"-2-dom",domain,".log",sep=""))
        table3 <- read.table(paste(location,prefix,"-3-dom",domain,".log",sep=""))
        table4 <- read.table(paste(location,prefix,"-4-dom",domain,".log",sep=""))
        table5 <- read.table(paste(location,prefix,"-5-dom",domain,".log",sep=""))

        #calcuate the average block time
        block_time=c(sum(table1[,8]),sum(table2[,8]),sum(table3[,8]),sum(table4[,8]),sum(table5[,8]))
        block_time_avg=mean(block_time)/4000000000

        #calculate the average cpu utalization time
        cpu_time=c(sum(table1[,4]),sum(table2[,4]),sum(table3[,4]),sum(table4[,4]),sum(table5[,4]))
        cpu_time_avg=mean(cpu_time)/4000000000

        #calculate the average wait time
        t=7200000000000
        wait_time=c(t-block_time[1]-cpu_time[1],t-block_time[2]-cpu_time[2],t-block_time[3]-cpu_time[3],
                        t-block_time[4]-cpu_time[4], t-block_time[5]-cpu_time[5])
        wait_time_avg=mean(wait_time)/4000000000

        #return a list with the (block percent, cpu percent, wait percent)
        return (c(block_time_avg/18,cpu_time_avg/18,wait_time_avg/18))
}
```

Bibliography

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[2] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. Comparison of the three CPU schedulers in Xen. *Special Interest Group on Operating Systems Operating Systems Review*, 35(2):42–51, 2007.

[3] Bryan Clark, Tod Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neefe Matthews. Xen and the art of repeated research, 2004. Online at `http://web2.clarkson.edu/class/cs644/xen/papers.html`.

[4] Intel Corporation. Moore's law, 2010. Online at `http://www.intel.com/technology/mooreslaw/`.

[5] Matthew DeDiana. An empirical performance evaluation of the xen sedf scheduler on multiple heterogeneous workloads. Allegheny College Senior Thesis CS08-01, 2008.

[6] Jim Elliott. The evolution of IBM mainframes and vm, 2004. Online at `http://www.linuxvm.org/Present/SHARE103/S9140jea.pdf`.

[7] Apache Software Foundation. Apache jmeter, 2009. Online at `http://jakarta.apache.org/jmeter/index.html`.

[8] Diwaker Gupta, Rob Gardner, and Ludmila Cherkasova. XenMon: QoS monitoring and performance profiling tool. 2005. Online at `http://www.hpl.hp.com/techreports/2005/HPL-2005-187.html`.

[9] Microsoft. Exchange server 2003 performance tools, 2006. Online at `http://technet.microsoft.com/en-us/library/aa996076(EXCHG.65).aspx`.

[10] Xen Project. Xen architecture overview. Online at `http://wiki.xensource.com/xenwiki/XenArchitecture?action=AttachFile&do=get&target=Xen+Architecture_Q1+2008.pdf`, 2008.

[11] Rami Rosen. Introduction to the Xen virtual machine, 2005. Online at `http://www.linuxjournal.com/article/8540`.

[12] Andrew S. Tanenbaum. *Modern Operating Systems*. Pearson Education, Inc., 3rd edition, 2009.

[13] Simon Tatham. Putty: A free telnet/ssh client, 2009. Online at `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`.

[14] VMware, Inc. Vmware milestones, 2010. Online at `http://www.vmware.com/company/mediaresource/milestones.html`.

[15] Xianghua Xu, Peipei Shan, Jian Wan, and Yucheng Jiang. Performance evaluation of the CPU Scheduler in Xen. In *Information Science and Engineering, 2008.*, volume 2, pages 68–72, Dec. 2008.

[16] Yaron. Credit-based cpu scheduler, 2007. Online at `http://wiki.xensource.com/xenwiki/CreditScheduler`.