PERFORMANCE EVALUATION AND CHARACTERIZATION OF VIRTUAL APPLIANCES

A Thesis Presented

by

Zhaoqian Chen

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements

for the degree of

Master of Science

in

Electrical and Computer Engineering

in the field of

Computer Engineering

Northeastern University

Boston, Massachusetts

January 2009
Acknowledgements

First and foremost, I would like to thank my academic and research advisor Professor David Kaeli, Director of the Northeastern University Computer Architecture Research Laboratory. His broad knowledge and logical thinking have been of great value for me. Without him, this thesis would not have been possible.

I also want to sincerely thank my fellow labmates in our research group. They have taught me a lot, helped me solve many problems and guided me during my research.

Lastly, and most importantly, I want to thank my beloved parents. They gave birth to me, raised me, and love me forever. I dedicate this thesis to them.
Abstract

System virtualization technology continues to increase in popularity across the datacenter. Independent Software Vendors (ISVs) are now using virtual machines to deliver software appliances to take advantage of management features provided by a virtualization environment. However, there is a growing need to better understand the performance of applications in this environment. New tools and approaches are needed to be able to model, evaluate and characterize software appliances when run on virtualized platforms.

To understand benchmarking in a server appliance environment, we select a commercial appliance, and use it to develop our approach to virtual appliance performance characterization. We begin with a study of AsteriskNOW, a SIP proxy server appliance. To explore this design space, we consider a range of virtual machine configurations. We also analyze the virtual appliance performance when the appliance scales both vertically and horizontally in the virtualized environment.

One of the outgrowths of this initial study is that we have developed a general
methodology for studying any virtualized appliance. We have designed and implemented VAmark, a performance characterization framework to collect and analyze the performance data available on virtual appliances. VAmark provides the capability to understand the behavior of the virtual appliance, quantify the associated performance interference, and identify the source of performance issues by profiling and analyzing system level statistics and hardware events.

To better understand the sharing of hardware resources in a virtual appliance environment, we utilize a number of standard workloads. We characterize SPECint2006 as a compute-intensive appliance, and both Apache Webserver and MySQL database server as typical server appliances. The VAmark framework can be used to evaluate and guide the deployment of virtual appliances for system integration.
Contents

Acknowledgements iii

Abstract iv

1 Introduction 1

2 Background 6

2.1 Virtualization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.2 Virtual Appliances . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.3 Popular Server Virtualization Platforms . . . . . . . . . . . . . . . . 11

2.3.1 VMware ESX Server . . . . . . . . . . . . . . . . . . . . . . . . 11

2.3.2 Xen Virtual Machine . . . . . . . . . . . . . . . . . . . . . . . . 12

2.4 Time Measurement Problem in Virtual Machine . . . . . . . . . . . . 13

2.5 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

2.5.1 Performance Analysis of Virtual Machines . . . . . . . . . . . . 15

2.5.2 Utilities for Performance Evaluation . . . . . . . . . . . . . . . . 16

2.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
### 3 Performance Evaluation of Virtual Appliance

3.1 Introduction .................................................. 19

3.1.1 An Overview of the SIP Protocol ...................... 20

3.2 Evaluation of AsteriskNOW on XenServer ............... 22

3.2.1 Evaluation Framework ................................. 22

3.2.2 Experimental Setup ................................. 24

3.2.3 Results and Analysis ................................. 26

3.3 Evaluation on VMware ESX Server ....................... 30

3.3.1 Experimental Setup ................................. 30

3.3.2 Results and Analysis ................................. 32

3.4 Summary of AsteriskNOW Performance Characterization 36

3.5 Appendix .................................................. 37

### 4 Performance Characterization Framework of Virtual Appliances

4.1 Introduction .................................................. 45

4.2 VAmark Framework ......................................... 46

4.3 VMmeter Design ........................................... 49

4.4 Experimental Setup ........................................ 50

4.5 Results and Analysis ..................................... 53

4.5.1 Performance Characterization ...................... 53

4.5.2 Performance Impact of Sharing PCPs ............... 55
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The VMware ESX Server Architecture</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Overview of Xen virtual machine architecture and I/O subsystem</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Basic stateful SIP scenario without authentication</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Our VA performance evaluation platforms</td>
<td>23</td>
</tr>
<tr>
<td>3.3</td>
<td>Integrated VA performance monitor and evaluation</td>
<td>23</td>
</tr>
<tr>
<td>3.4</td>
<td>The XenServer Architecture</td>
<td>24</td>
</tr>
<tr>
<td>3.5</td>
<td>Performance comparison under different configurations</td>
<td>30</td>
</tr>
<tr>
<td>3.6</td>
<td>Performance of vertical scalability</td>
<td>33</td>
</tr>
<tr>
<td>3.7</td>
<td>Physical CPU utilization with CPU affinity</td>
<td>33</td>
</tr>
<tr>
<td>3.8</td>
<td>Performance of horizontal scalability</td>
<td>35</td>
</tr>
<tr>
<td>3.9</td>
<td>Physical CPU utilization with CPU affinity</td>
<td>35</td>
</tr>
<tr>
<td>3.10</td>
<td>Statistics of [1 Asterisk VM with 1 VCPU]</td>
<td>38</td>
</tr>
<tr>
<td>3.11</td>
<td>Statistics of [1 Asterisk VM with 2 VCPUs]</td>
<td>39</td>
</tr>
<tr>
<td>3.12</td>
<td>Statistics of [2 VMs on CPU competition, 1 VCPU per VM]</td>
<td>40</td>
</tr>
<tr>
<td>3.13</td>
<td>Statistics of [2 VMs on CPU competition, 2 VCPUs per VM]</td>
<td>41</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Different VM configurations considered for the dual-core environment.</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>Performance comparison of nttcp benchmark.</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Different VM configurations used in our scalability study.</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>The current benchmarking elements present in VMmeter.</td>
<td>50</td>
</tr>
<tr>
<td>4.2</td>
<td>Summary of our experimental VA platform.</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>Resources utilization of running gcc vs. VMmeter network I/O test.</td>
<td>54</td>
</tr>
<tr>
<td>4.4</td>
<td>Resource utilization of MySQL vs. VMmeter disk I/O test.</td>
<td>55</td>
</tr>
<tr>
<td>4.5</td>
<td>VM-level Statistics of gcc vs. the VMmeter CPU test.</td>
<td>57</td>
</tr>
<tr>
<td>4.6</td>
<td>Summary of CPU busytime of running MySQL.</td>
<td>59</td>
</tr>
<tr>
<td>4.7</td>
<td>Summary of L2 cache misses for MySQL.</td>
<td>59</td>
</tr>
<tr>
<td>4.8</td>
<td>Summary of ITLB misses of running MySQL.</td>
<td>60</td>
</tr>
<tr>
<td>4.9</td>
<td>Summary of CPU busytime when running gcc.</td>
<td>61</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

Today, virtualization technology plays a critical role in many computing enterprises and datacenters. Virtualization can provide lower cost of ownership, higher performance/watt, and improved cost/performance. A virtual machine monitor (VMM) or hypervisor is a software layer that virtualizes hardware resources, abstracting away the details of the actual underlying physical hardware. The concept of the virtual machine was invented by IBM in the 1970’s to enable multiple virtual machines to run concurrently on expensive mainframe hardware [12]. Its recent rise in popularity is tied to its ability to share hardware infrastructure across multiple virtual machines run on commodity hardware.

Virtualization has been used effectively for server consolidation and optimization of hardware resources. Server consolidation helps enterprises achieve higher system utilization, reduce management complexity and total cost. Current popular virtualization products include Citrix’s XenServer [16], Microsoft Hyper-V [24] and VMware
To support efficient execution on commodity hardware, a number of hardware assists have been developed such as Intel’s VT-x [28] and AMD’s SVM [13]. These hardware enhancements illustrate that virtualization is a technology that is here to stay, and these features have helped to accelerate the move to deploying virtual machines (VMs) in production software appliances (i.e., virtual appliances or VAs).

However, when a virtual appliance runs in server consolidation mode, the associated performance interference from other virtual appliances on the same physical hardware platform cannot be ignored. We need to understand the performance characteristics of the virtual appliance in order to predict how the appliance will perform when competing workloads share elements of the same virtualized environment.

Prior performance studies have focused on specific types of virtualized workloads, including: server consolidation workloads [6, 31] and I/O intensive disk workloads [9, 5]. But a general approach or framework to model, evaluate and characterize software appliances when run on virtualized platforms has not been developed.

In our work, we first start with a performance evaluation work of AsteriskNOW, a popular SIP proxy server appliance running in a virtualized environment. Our goal is to develop an understanding of how best to characterize this rather complex voice-over-IP server appliance, before we attempt to arrive at a set of generalized
performance tools. This phase of our work utilizes a XenServer virtual machine environment and is run on an Intel dual-core Xeon CPU. We characterize the performance of the Asterisk appliance and quantify the virtualization overhead for various VCPU (i.e., virtual CPU) configurations. We also consider and analyze the performance impact of running the appliance alongside a competitor VM running either CPU, networking or disk workloads.

Once we developed our VA evaluation framework, we moved to an Intel quad-core system so we could further explore issues related to the number of physical CPU cores in the system. To also consider the impact of the VMM chosen, we moved our study to VMware’s ESX Server.

A major output of this thesis is the design and development of VAmark, a performance characterization framework providing the capability to understand the behavior of the virtual appliance. VAmark enables us to quantify performance interference and locate performance bottlenecks in a virtual appliance environment.

As part of this thesis we develop a new metric called the normalized performance score. We use this metric to quantify the performance interference induced by resource competition. We developed VMmeter as a standard competitor VM to generate workloads on CPU, memory, network and disk I/O. VMmeter can be configured to compete for the hardware resources and impact the performance of the virtual appliance under test.
To collect and analyze the runtime details, our VAmark framework provides profiling tools that allow us to gain a better understanding of the performance hit of the virtual appliance. VAmark is a guide to decide how best to provision the available hardware resources and the deployment of virtual appliances for system integration.

The main contributions of this thesis include:

- we present a comprehensive performance evaluation study of AsteriskNOW virtual appliance, consider different VM configurations, hardware configurations, and VMM implementations,

- we introduced the concept of a normalized performance score to quantify the performance interference when running multiple virtual appliances,

- we developed VMmeter as a standard competitor virtual machine. It runs different benchmarking tests to generate workloads and create resources competition, and

- we designed the VAmark framework and implemented it; we evaluated and analyzed the performance of three different workloads running in a virtual appliance environment: SPECint 2006 benchmarks [30], MySQL database server [4] and Apache web server [14].

The rest of this thesis is organized as follows. Chapter 2 provides background information on virtualization technology and virtual appliances. Related research
focused on performance evaluation on virtualized systems is surveyed. Chapter 3 describes our performance evaluation study of the AsteriskNOW appliance on both XenServer and VMware ESX Server environments. Chapter 4 presents VAmark, our new performance characterization framework of virtual appliances. We show characterization results and performance analysis of several virtual appliances in this chapter. Finally, Chapter 5 draws conclusions and presents the future directions for research.
Chapter 2

Background

This chapter provides an overview of virtualization technology. Section 2.1 introduces the concept and features of virtualization, highlighting the problem of performance isolation in a virtualized environment, which is directly addressed in this thesis. Section 2.2 describes two popular server virtualization platforms: VMware ESX Server and Xen virtual machine. Section 2.3 describes the benefits of the virtual appliance environment. Section 2.4 provides a review of the current literature on performance evaluation of virtual machines.

2.1 Virtualization

The virtual machine monitor (VMM) or hypervisor is a software layer that virtualizes hardware resources, abstracting away the details of the actual underlying physical hardware. There are two main approaches to server virtualization. Full virtualization (e.g., VMware ESX Server) allows an unmodified operating system to run on top of
a hypervisor without any awareness of the underlying hypervisor layer. *Paravirtualization* runs a modified operating system that is fully aware of the hypervisor and cooperates in the virtualization process.

The complexity and performance of each virtualization approach can vary [19]. Usually, paravirtualization is more efficient than full virtualization; full virtualization has gained popularity for running proprietary operating systems and microprocessor vendors have begun to provide instruction set support to accelerate this style of virtualization. Full virtualization can demonstrate superior performance for workloads that are IO dominant.

When enterprises deploy virtualization, a number of associated features such as seamless system migration, system availability and ease in power cycling improve the efficiency of datacenter management. Virtualization technology also provides strong fault isolation and security isolation that prevent a compromised application or operating system from bringing down the entire platform.

Performance isolation is another potential goal of virtualization. It would be nice if the resource consumption of one VM would not impact the performance of other VMs. However, current virtualization technologies are not able to provide effective performance isolation across VMs [33, 31]. If one VM is using shared resources, other VMs can suffer in terms of performance due to contention for the same hardware resource. The concept of *performance interference* in this environment is discussed.
Virtualization hides one VM from another, so a single guest operating system (a "guest" is another term used for a virtual machine) can not communicate with other guest operating systems. While the hosting VMM is responsible for managing the underlying hardware resources (as most operating systems do), the VMM has limited visibility into the guest OS and the applications running on top of it, which makes optimization in the VMM difficult. Furthermore, some VMM implementations contain layers when accessing disk storage and network resources. For instance, the Xen hypervisor routes all I/O operations from all VMs through a device driver domain. When running I/O intensive workloads, the single domain may become a bottleneck in the system.

When virtualization is deployed, enterprises pay less attention to how their applications are performing in this virtualized environment. When a virtualized appliance runs on the same hardware platform with other virtualized appliances, we need to pay careful attention to the associated performance impact. For some near real-time applications (e.g., telephony servers, game servers, etc.), performance and latency are crucial and must be managed carefully.

While some commercial studies have compared their systems to competing virtualized environments, little has been done by the performance evaluation community to analyze the performance and scalability of these environments.
2.2 Virtual Appliances

Software appliances are developed by Independent Software Vendors (ISVs) and provided for software applications to be pre-installed and pre-configured. A software appliance generally includes a customized and optimized operating system and the software application packaged within it. A virtual appliance is defined as a minimal virtual machine image that contains the software appliance designed to run in a virtualized environment. ISVs take advantage of producing virtual appliances from several aspects:

- Simple deployment for ISVs: virtual appliances eliminate complex and expensive installation and configuration processes.

- Added security: virtual appliances run on slim OSs that only include the program and components required to support the specific applications. The OS occupies a smaller footprint compared to general purpose OSs and provides fewer features, and thus is less vulnerable.

- Portability: virtual appliances deliver flexibility, providing vendor independence and allowing users to select the platform that meets their needs.

- Simpler IT management: users can easily deploy management services on virtual appliances such as system backup and checkpointing, and load balancing.
While virtual appliances provide many benefits, performance management in this environment still remains a poorly understood area. The need of performance evaluation and analysis on virtual appliances continues to grow for different groups of people:

- **IT Administrator**: When a new ISV demands resources to run their application, virtualization addresses the problem by providing a VM with virtual resources that run on a large multi-processor platform. But ISV is however unaware of any virtualization layer. The customers may expect much better performance on the new platform without any concern for the fact that the applications are running in a virtualized environment. Hence, IT administrators have to ensure that all applications running in the virtualized environment are perform well and get their fair share of the resources.

- **Platform Designers**: Evaluation of commercial server workloads is challenging since it not only requires the understanding of each individual server application, but also the behavior of how these applications are affected by virtualization overheads, how they are affected when they run in consolidation mode with other workloads, how they share resources within the platform and finally how they scale in terms of future platform requirements.

- **VMM/hypervisor Developers**: To provide appropriate feedback to improve VM schedulers and help manage the resources, it is important to characterize and
analyze the execution profiles of server workloads.

New tools and approaches are required to be able to model, evaluate and characterize virtual appliances. The described framework in Chapter 4 allows us to evaluate and characterize the performance interference caused by virtual appliances.

2.3 Popular Server Virtualization Platforms

2.3.1 VMware ESX Server

VMware ESX Server is a platform virtualization software developed by VMware Inc. Figure 2.1 shows the VMware ESX Server architecture.

VMware ESX Server inspects all I/O from virtual machines. Disks and networks interfaces are presented as virtual devices in the virtual machine. These virtual devices are generic to the guest operating system, regardless of the underlying physical hardware. An I/O request issued by the guest OS first goes to the driver in the virtual machine. The virtual device driver then turns the I/O request into an access...
to the I/O ports to communicate to the virtual devices using privileged x86 IN/OUT instructions. These instructions are trapped by the virtual machine monitor (VMM), and then handled by device emulation code in the VMM, based on the specific I/O port being accessed. The VMM next calls the device-independent network or disk code to process the I/O. For disk I/O, ESX server maintains a queue of pending requests per virtual machine for each target SCSI device. The I/O requests are then sent down to the device driver loaded into ESX Server for the specific physical device.

### 2.3.2 Xen Virtual Machine

Xen is an open-source virtual machine monitor that allows multiple guest operating system instances to run concurrently on a single physical machine [7]. Xen supports both para-virtualization and full virtualization.

![Figure 2.2: Overview of Xen virtual machine architecture and I/O subsystem.](image)

Figure 2.2 shows the architecture and the I/O communication structure present in Xen 3.0. The lowest and most privileged layer is the Xen hypervisor (i.e., VMM),
which presents a virtual hardware abstraction to the guest VMs. VMs are scheduled by the Xen hypervisor to make effective use of the available physical CPUs and memory. Each application running in a guest OS accesses hardware devices through a special privileged VM called Domain-0. This privileged VM directly talks to the I/O device drivers. All other VMs indirectly use a simple device driver that communicates with Domain-0’s virtual device to access real hardware devices. Domain-0 then maps through bridges and routes each physical interface.

Under para-virtualization, modified guest OSs use *hypercalls* to issue privileged operations (e.g., updating page tables) to the Xen hypervisor. A hypercall is a software trap from a domain to the hypervisor and the Xen hypervisor notifies the guest OS after the request has completed [7].

### 2.4 Time Measurement Problem in Virtual Machine

Time measurement is an important factor when we evaluate the performance of any virtual appliance. We want the ability to accurately measure the time elapsed as a performance metric. Some appliances may utilize time to decide the behavior in execution (e.g. timeouts for servers, transfer rates, and etc.). However, because virtual machines work by timesharing host physical hardware, a virtual machine cannot exactly duplicate the timing behavior of a physical machine.

A fully virtualized system, just like an OS running on bare hardware, relies on
the timer interrupt for its timekeeping. This means that an idle virtual machine still has to process hundreds of interrupts a second and missed interrupts result in inaccurate time. Paravirtualization keeps a stable time in the hypervisor, having the guest OS query the hypervisor time. This also allows the hypervisor to mask timer interrupts to paravirtualized guests when they are idle. It simply tells the hypervisor “wake me up in half a second” (or whenever the next scheduled event is), and goes to sleep. This means that idle guests use less CPU time, allowing active guests to run on the same physical hardware. Having idle virtual machines physically idle also allows the hypervisor to put the CPU in powersaving mode when nothing is running. A few hundred interrupts per second to a few dozen virtual machines would prevent powersaving from working properly.

The amount of time the virtual machine slowed down when the guest operating system was ready to run but the virtual machine was descheduled by the host scheduler is called “stolen time”. VMware allows many timer devices in a virtual machine to fall behind real time and catch up as needed, yet remain sufficiently consistent with one another that software running in the virtual machine is not disrupted by anomalous time readings. Time measurements taken within a virtual machine can be somewhat inaccurate because of the difficulty of making the guest operating system clock keep exact time. When using a paravirtualized timer device, the guest operating system is allowed to explicitly account for “stolen time”.

Because of these issues, total CPU load (or, conversely, total idle time) measured from within a virtual machine is not a very reliable number. Therefore, if software run in a virtual machine that measures and adapts to total system load, the software’s measurement and adaptation algorithms may need to be modified. The guest operating system does not have the full view of what is happening on the whole virtualization platform. To get a complete picture, testers need to look at statistics taken by the VMM, such as esxtop on ESX and xenmon on Xen. Another solution is to use a network time server for time measurement. For example, the National Institute of Standards and Technology (NIST) provides a public time service to query the standard time server and retrieve accurate time.

2.5 Related Work

2.5.1 Performance Analysis of Virtual Machines

With virtualization rapidly regaining relevance in the last few years, there have been a number of recent papers on the performance overhead associated with virtualization and the analysis of specific aspects of virtualization.

Apparao et al. [6] studied the performance slowdown of a server consolidation benchmark on a multi-core platform. They looked at architectural characteristics such as CPI (cycles per instruction) and L2 misses per instruction and analyzed the benefits of larger caches for the consolidated workload.

Koh et al. [33] collected performance metrics and runtime characteristics using
an instrumented Xen hypervisor. From subsequent analysis of the performance data, they clustered the application and developed mathematical models to predict the performance of application from its workload characteristics.

Ahmad [5] presents an efficient implementation of disk I/O workload characterization using online histograms in the VMware ESX Server. This technique allows transparent and online collection of disk I/O performance metrics.

Chadha et al. [9] developed an execution driven simulation based analysis methodology to evaluate the performance of virtualized workload. This methodology provides detailed information at the architectural level and evaluates the potential hardware enhancements to reduce virtualization overhead. They applied this methodology to study the network I/O performance of Xen in a full system simulation environment.

2.5.2 Utilities for Performance Evaluation

System administrators or testers manage the virtualization platform through a service console or privileged VM (e.g., Domain-0 in Xen). But these system do not have direct access to the underlying hardware so that traditional system utilities (e.g., top or sar in Unix) can not capture the status of all VMs on the platform. Testers have to access the VMM/hypervisor to retrieve the resource utilization of VMs. Normally, virtual machine developers add APIs in the VMM and allow others to build specific utilities for monitoring virtual machines. Many efforts are developing utilities for virtualized environments to assess performance on these platforms.
For example, *xentop* is a utility similar to the Unix *top* command that displays information about the Xen system. Xentop collects information dynamically and keeps updating system status. It captures CPU usage, network I/O throughput, and disk I/O counts for each domain. We use xentop in this thesis to collect VM-level statistics.

*Xenoprof* [23] is a system-wide profiler capable of profiling the Xen hypervisor, multiple Linux guest operating systems, and applications. Xenoprof is modeled after the OProfile [1] profiling tool available on Linux systems. Xenoprof allows multiple guest systems to use and share the performance counters that are available on X86 CPUs. In the *xenoprof* work, the authors provide a case study on network applications in order to show how to diagnose bugs and performance problems on Xen.

To evaluate the potential virtualization power of the platform, many organizations have been working on a standardized consolidation benchmarking suite. *VMmark* [21] was developed by VMware and is a tile-based benchmark consisting of several workloads running simultaneously in separate virtual machines. Each workload component is based upon a single system-level benchmark running with full utilization. The performance of each workload is measured and used to form an aggregate score. The scores generated when running multiple tiles simultaneously are summed to create the overall benchmark score. VMmark is mainly used for platform/hardware capability
measurement. The SPEC organization is working on a new benchmark for virtualization. The SPEC Virtualization Committee [11] plans to deliver a benchmark that will model server consolidation of commonly virtualized systems. The goal is to provide a means to fairly compare server performance while running multiple virtual machines. This new benchmark is not particularly suitable for the analysis of virtual appliances.

2.6 Summary

In this chapter, we described the basis of virtualization technology and introduced two popular server virtualization platforms: VMware ESX Server and Xen Virtual Machine. We presented the performance isolation issues and the challenges of measuring performance when utilizing virtualization to run concurrent software appliances. Our motivation and approach for evaluating and characterizing virtual appliances was then discussed. Finally, we provided a review of some of the recent research on evaluation and characterization of virtual appliances. In the next chapter, we will describe our evaluation work on the AsteriskNOW virtual appliance.
Chapter 3

Performance Evaluation of Virtual Appliance

3.1 Introduction

To begin this thesis we select a target virtual appliance that will help us understand all of the challenges involved in benchmarking on a virtualization environment. The software appliance we selected is AsteriskNOW [18], which is a well-known telephony server appliance running the SIP protocol (described later in this section).

In the next section, we present our evaluation work of the AsteriskNOW appliance on XenServer platform. This study is carried out on an Intel dual-core system. We show results and analysis of the AsteriskNOW appliance under various VM configurations. From the analysis, we found that limited number of physical CPUs is a major obstacle for performance.

We then move to an Intel dual-processor dual-core system. We also study performance on a second virtualization environment. The scalability of the AsteriskNOW
appliance is analyzed based on the VMware ESX Server platform. We show the performance of the platform under different VM scaling approaches to determine the best configuration for maximum appliance throughput.

### 3.1.1 An Overview of the SIP Protocol

The Session Initiation Protocol (SIP) [32] is an application layer control protocol for creating, maintaining, and tearing down sessions for various types of media, including voice, video, and text. SIP is growing in importance as it is being used for many media-oriented applications such as Voice-over-IP (VoIP), voicemail, instant messaging, IPTV, network gaming, and more. It is also the core protocol for the IP Multimedia Subsystem, the basis for the 3rd-Generation Partnership Program for both fixed and wireless telephone networks. SIP relies on an infrastructure of servers, which are responsible for maintaining the location of users and forwarding SIP messages across the application-layer SIP routing infrastructure toward their eventual destinations. The performance of these SIP servers is thus crucial to the operation of the infrastructure, as they can have a primary impact on the latency of media applications (e.g., for initiating a phone call). Service providers clearly require performance information to understand how to provision their infrastructure to provide reasonable QoS.

Figure 3.1 shows a basic SIP scenario without authentication. The SIP client wishes to establish a session via the server and sends an INVITE message to the
server. The server responds with a TRYING message to inform the client that the message has been received. The server generates an OK message, which is sent to the client. The client then replies an acknowledgment message. The session is now established. When the conversation is finished, the client user hangs up and generates a BYE message. The server side responds with an OK to the user.

Performance is measured by clients using the SIPp workload generator [15]. SIPp provides for a wide range of SIP scenarios to be tested, such as user-agent clients (UAC), user-agent servers (UAS) and third-party call control. SIPp is also extensible by writing third-party XML scripts that define new call flows. SIPp has many run-time options, such as multiple transport (UDP/TCP/TLS) support and MD5-based hash digest authentication.
3.2 Evaluation of AsteriskNOW on XenServer

3.2.1 Evaluation Framework

Our Virtual Appliance (VA) performance evaluation framework is shown in Figure 3.2. This framework consists of two separate systems: 1) the VA tester/monitor and 2) the target platform (including the Virtual Appliances and VMM) under test. The VA tester/monitor software contains a VMM-level data profiler, client-based VA workload drivers to drive each VA, and a management user interface (UI). The VMM profiler probes the VMM to provide virtual machine information, such as VM status, virtual/physical CPU utilization, VMM scheduling statistics, virtual I/O speed, etc.

A number of different VA workloads are configured on the target system to act as typical virtual appliances. The VA tester/monitor drives the different VAs with load. We then obtain VMM statistics and compile these back on the VA tester/monitor system. Using these workloads and the VMM profiler together, we are able to analyze each VA’s performance. The management interface is designed to allow the tester/monitor system administrator to monitor VA performance.

Our VA test/monitor is presently configured on a separate system in order to eliminate any overhead. The VA tester/monitor could be integrated into the platform to reside on a privileged VM, as shown in Figure 3.3. This privileged VM has full access to the VMM so that our VA tester/monitor is still able to profile VMM stats and all other VMs. Thus, future platform vendors will be able to produce a single
Figure 3.2: Our VA performance evaluation platforms.

box with VAs installed and with the VA performance tester/monitor integrated.

Figure 3.3: Integrated VA performance monitor and evaluation.

We have implemented this evaluation framework on XenServer virtual machine (see Figure 3.4) which is an enterprise version of Xen from Citrix [16]. XenServer is based on the open source Xen hypervisor, which delivers low overhead and near-native performance. Leveraging hardware assisted virtualization, XenServer delivers faster,
efficient virtualization computing. It opens APIs to allow users to access and control advanced functions from their existing server and storage hardware. XenServer consists of the Xen hypervisor, management software, and a number of service features (P2V conversion, migration, built-in guest templates and etc.). The Xen hypervisor profiling tool we created is written in Python and utilizes the XenAPI [17] to collect VMM statistics (e.g., CPU utilization, network interface speed). The sampling granularity is adjustable.

3.2.2 Experimental Setup

The hardware used in our initial set of experiments is an Intel Dual-Core CPU Xeon 3060 that supports Intel VT [28], running at 2.4GHz with 4MB L2 Cache and 4GB main memory.

Two VMs are setup for all of our experiments. We instantiate either two identical
Asterisk VMs, or one Asterisk VM and one “Competitor VM” (described later). Each VM is assigned 1GB of memory and sufficient disk space. The VM configurations being tested are shown in Table 3.1.

<table>
<thead>
<tr>
<th>No.</th>
<th># of VM</th>
<th>VCPU/VM</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Only Asterisk VM</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Only Asterisk VM</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>Asterisk with CPU competition</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>Asterisk with CPU competition</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>Asterisk with Network competition</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>Asterisk with Disk competition</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1</td>
<td>Two Asterisk VM running concurrently</td>
</tr>
</tbody>
</table>

Table 3.1: Different VM configurations considered for the dual-core environment.

We are using AsteriskNOW version 1.0 (we will refer to this VM as the “Asterisk VM”) running under hardware-assisted Virtual Machine mode. AsteriskNOW includes a copy of the rPath Linux operating system [2] and the Asterisk telephony server application. A Debian Etch (4.0) Linux operating system created from XenServer’s built-in templates pool is run on another VM (we label this second VM as the “Competitor VM”). We run a number of workloads on the Competitor VM, each stressing a different hardware resource.

Super Pi [3] is a CPU-intensive workload that calculates Pi to a specified number of digits after the decimal point. Nttcp [26] is a network-intensive workload that performs TCP throughput testing. This workload starts a network connection between two systems and then transfers data. Both sides measure the time and number of bytes transferred. IOzone [29] is a disk-intensive workload that generates and
measures a variety of file operations with both random and sequential operations.

We run the SIPp workload from another independent computer system, which send packets over a 100Mb Ethernet. The SIPp program is configured as a UAC client that sets up a socket and employs the default UDP port 5060 to send multiple messages to the server. The SIP scenario is a stateful proxying without authentication. This configuration can achieve fairly good performance [27]. We are interested not only in the maximum capacity of the server (i.e., throughput), but also how the application behavior degrades under excessive load. We gradually increase the workload (calls per second), making a slow transition from non-overloaded to overloaded status.

3.2.3 Results and Analysis

The statistics collected by our profiling tools include physical CPU (PCPU) utilization, virtual CPU (VCPU) utilization for all VMs, and physical network (PIF) I/O speed. The SIPp benchmark provides statistics about the number of successful/failed calls, response time distributions, and retransmission statistics.

Modeling a Single Asterisk VM

Figure 3.10 shows the resulting performance of running the Asterisk VM with only one VCPU. We find that this VCPU gets pinned to a single physical CPU by the Xen VM scheduler. The other physical CPU is used by Domain-0 to handle I/O processing. Running in a stable (non-overloaded) environment, the total network speed is around 2MB/s. We have found that when the Asterisk server reaches the
maximum throughput, the CPU usage of the Asterisk application is about 70%. After the server is overloaded, it fails to process calls in time and starts sending out RTP traffic indicating that the server is currently too busy to respond. This impacts network traffic severely.

The CPU usage of the Asterisk VM decreases due to less call processing work to perform. But the CPU used by Domain-0 is impacted since it is encountering extra network traffic. For our experimental environment, the maximum throughput is around 700 calls per second (CPS).

When the Asterisk VM is assigned 2 VCPUs, we see the impact on the two physical CPUs which will have almost the exact same utilization, as shown in Figure 3.11. This means the total workload is being balanced across both CPUs by the Xen VM scheduler. The two virtual CPUs of the Asterisk VM are floating on the two physical CPUs. Since there are only two physical CPU cores on the system, the VCPUs of the Asterisk VM compete with the VCPU of Domain-0. From the results shown in Figure 3.11e, the performance (maximum throughput) is 600 CPS which is even lower than the previous configuration (1 VCPU for the Asterisk VM). The overhead of frequent CPU swapping and scheduling degrades call throughput by as much as 100 CPS.
Two Virtual Machines

1) One Asterisk VM and One Competitor VM

Next we consider configuring two VMs. We assign a VCPU to each VM. Since two VMs are competing for hardware resources (i.e., shared memory and buses), the Asterisk application fails to achieve the same performance as the single-instance case of Asterisk (see Figure 3.12). The performance of the Competitor VM running the SuperPi program is affected by the Asterisk VM as well. The total execution time of 231.4 seconds for $\pi$ computed to $2^{22}$ digits in a non-virtualized environment, versus the Super Pi program spends 253.8 seconds in the Competitor VM, experiencing a 10% execution slowdown.

When we assign two VCPUs to each VM, the performance degrades even further and the maximum throughput is only about 100 CPS (see Figure 3.13). Since we have only a single dual-core CPU, the Xen VMM must virtualize a total of six VCPUs (each of the two VMs and Domain-0 each have two VCPU). Creating more VCPUs without considering the number of PCPUs can lead to poor performance.

Next, we configure the system so each VM has a single VCPU. We run the network workload nttcp in the Competitor VM. We use another independent physical machine to transfer data to the Competitor VM on the network while Asterisk is processing calls. This network-hungry Competitor VM competes for network bandwidth with the Asterisk VM. Table 3.2 shows the performance of running the nttcp workload. In a
virtualized environment, it only introduces a 5% overhead. The maximum throughput of the Asterisk VM is 650 CPS (see Figure 3.14), which is not far from the baseline (1 VM with 1 VCPU). The total network traffic is about 12 MB/s. From the previous statistics, we know that the network traffic just from the Asterisk VM is 2 MB/s, which means that network is not a limiting factor for the Asterisk VM.

<table>
<thead>
<tr>
<th></th>
<th>Real (sec)</th>
<th>CPU (sec)</th>
<th>Real (Mbit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Virtualized</td>
<td>313.25</td>
<td>2.53</td>
<td>94.15</td>
</tr>
<tr>
<td>Virtualized</td>
<td>329.32</td>
<td>2.63</td>
<td>89.55</td>
</tr>
</tbody>
</table>

Table 3.2: Performance comparison of nttcp benchmark.

To better understand how Asterisk competes for disk I/O, we ran the IOzone workload in the Competitor VM. We configured the system identically to the last experiment, with the exception of replacing nttcp with IOzone. The IOzone workload obtains the same disk performance in both virtualized and non-virtualized environments. The overhead of disk latency is about 2% (378.85s vs. 385.35s). The maximum performance of the Asterisk VM is about 480 CPS.

2) Two identical Asterisk Virtual Machines

To complete this analysis space, we ran two Asterisk VM at the same time. From our testing results, we found that the system behavior and performance of two VMs are almost identical. The workload reaches a maximum throughput at 800 CPS as shown in Figure 3.16).

Figure 3.5 compares of all the configurations we have tested. These experimental
results tell us that for multiple CPU-intensive applications running on a virtualized platform, sharing CPUs introduces significant overhead and degrades the performance. One solution is to provide multiple CPUs so that each VM possesses its own VCPUs.

### 3.3 Evaluation on VMware ESX Server

#### 3.3.1 Experimental Setup

Next, we move to a VMware ESX Server environment to continue our study of AsteriskNOW. The system used in the following experiments contains an Intel dual-processor dual-core Xeon CPU that also supports Intel VT [28], running at 2.3GHz with 4MB L2 Cache and 4GB main memory.

The goal of this stage of our study is to understand the performance scalability in a virtualized environment. Our present dual-core environment was somewhat limiting
in this respect. We are interested to learn how the AsteriskNOW appliance “scales” in the virtualized environment. We use the term *Vertical Scalability* referring to the adding more virtual CPUs to a VM. This form of scaling employs only one instance (VM) with a multi-core CPU. The term *Horizontal Scalability* refers to creating more VMs on the platform. Horizontal scaling typically implies multiple instances (VM), residing on the host server. Naturally, one solution does not address all environments. Many times, blends of these two approaches are employed often.

When we instantiate multiple identical Asterisk VMs, each Asterisk VM is assigned with 512MB of main memory and sufficient disk space. VCPU settings are changed with different tests shown in Table 3.3.

<table>
<thead>
<tr>
<th>No.</th>
<th># of Asterisk VM</th>
<th>VCPU/VM</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Horizontal Scalability:</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Single Asterisk VM</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>Different number of VCPUs</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>Vertical Scalability:</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>Multiple Asterisk VM</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>4</td>
<td>Maximize Platform Usage</td>
</tr>
</tbody>
</table>

Table 3.3: Different VM configurations used in our scalability study.

The control over the subset of physical processors each virtual machine can use is called *CPU affinity*. The default setting is to use no affinity, and this is usually the best choice for most situations. However, if we encounter a resource-hungry virtual machine running on a host server, it will be beneficial to use CPU affinity to isolate the intensive virtual machine and provide the VA with reasonable performance. Doing
so will also protect the performance of all other virtual machines running on that same host server by changing each of their affinity settings to a different processor from that of the resource-intensive virtual machine. Thus, in our experiments, for each configuration, we study the performance with two CPU affinity options: CPU affinity or no CPU affinity.

The scalability study is carried on VMware ESX Server 3.1. For the best networking performance, we setup vmxnet virtual network device on the ESX Server. The vmxnet driver installed from VMware Tools implements an idealized network interface that passes through network traffic from the virtual machine to the physical cards with minimal overhead.

The SIP configuration is configured to be the same as in the previous experiments. We run the SIPp program from another independent computer system sending workload to the Asterisk VM(s). The ESX Server and workload generator machine communicate via a gigabit switch so as to maximize the performance. We measure the total throughput (maximum capacity) of the server running multiple Asterisk VMs.

3.3.2 Results and Analysis

Vertical Scalability

Figure 3.6 shows the performance of a single Asterisk VM scaling vertically. The average performance improvement over 1 VCPU configuration (1000 CPS) is about 20% for 2 VCPUs configuration (1200 CPS) and 35% for 4 VCPUs configuration (1350
CPS). Using CPU affinity results in a performance improvement around 5%-10%, since it reduces overhead of scheduling VCPUs on all PCPUs. The only exception is under 4 VCPUs configuration. Since the 4 VCPUs are always scheduled on all 4 PCPUs (independently of whether CPU affinity is set or not), this does not make a difference for the performance of Asterisk appliance.

Figure 3.6: Performance of vertical scalability.

Figure 3.7: Physical CPU utilization with CPU affinity.
We examine the PCPU utilization for the range of configurations (that also consider CPU affinity) shown in Figure 3.7. For the 1 VCPU configuration, the Asterisk VM is pinned to PCPU3. With single VCPU, the VM reaches a maximum at 85% of CPU utilization. PCPU0 is always used for VMware services (e.g., management console) which take about 10% CPU utilization. For the 2 VCPUs configuration, the Asterisk VM is pinned to PCPU2 and PCPU3. When VM is running on 2 PCPUs, the utilization is well averaged by the scheduler about 60%. Similarly, using the 4 PCPUs configuration, the workload is well balanced on 4 PCPUs with 40% utilization. But the average utilization of 4 PCPUs is lower than the PCPU utilization in the single PCPU scenario. This implies that the Asterisk appliance does not fully take advantage of 4 CPUs and the server is underutilized. Server consolidation by running multiple Asterisk VMs is needed to enhance the total throughput.

**Horizontal Scalability**

Figure 3.6 shows how the performance of the Asterisk VMs scale horizontally. The baseline configuration is “1 VM with 4 VCPUs” that produces 1350 CPS. The configuration “2 VM, 2 VCPUs per VM” rises 70% to the baseline (2400 CPS). The configuration “4 VM, 1 VCPU per VM” improves 150% to a maximum throughput of 3400 CPS. Enabling CPU affinity also improves performance by about 10%. However, in the “4 VM, 4 VCPU per VM” configuration, the total throughput is very low, only about 1600 CPS. The reasons of this problem are due to: First, according to the
previous vertical scalability study, the Asterisk appliance does not scale well when moving to multiple CPU systems. Second, there are a total of 16 VCPUs virtualized on the platform. The hypervisor is responsible to dynamically map 16 VCPUs to 4 PCPUs which increases a lot overhead by the VMM layer. Lastly, cache and core contention also result in performance loss.

Figure 3.8: Performance of horizontal scalability.

Figure 3.9: Physical CPU utilization with CPU affinity.
The PCPU utilization when considering horizontal scalability is shown in Figure 3.9. Under the “1 VM with 4 VCPUs” and “4 VM, 4 VCPUs per VM” configurations, each Asterisk VM is allowed to run on all 4 PCPUs. In the “2 VM, 2 VCPUs per VM” test, one Asterisk VM is pinned to PCPU0 and PCPU1; the other Asterisk VM is pinned to PCPU2 and PCPU3. In the “4 VM, 1 VCPU per VM” test, each Asterisk VM is configured to exclusively own a PCPU. Setting CPU affinity reduces overhead during scheduling. The utilization of CPU resources is better balanced on 4 PCPUs. And the average usage grows linearly with the performance. The maximum CPU utilization is reached at about 80% per PCPU core, achieving the maximum total throughput.

3.4 Summary of AsteriskNOW Performance Characterization

In this chapter, we explored how best to configure a VA performance analysis framework to study VA performance and scalability. We also presented a performance evaluation of AsteriskNOW, a SIP proxy server appliance. We created an environment based on the XenServer virtualization platform and evaluated the performance of AsteriskNOW appliance. We also considered a range of configurations when studying performance. From our experimental results, we find that the Xen VMM manages hardware resource utilization (i.e., disk and network I/O) effectively across VMs. But performance varies wildly when running multiple virtual appliances at the same time.
Sharing CPUs among VMs can also significantly impact performance.

We also evaluated scalability on a VMware ESX environment on a multi-processor multi-core system, in order to consider environments that run multiple virtual appliances concurrently. Our experimental results show that both horizontal and vertical scaling of the AsteriskNOW appliance can improve the performance (maximum throughput), though the Asterisk appliance does not fully take advantage of a multi-core system. Virtualizing more VCPUs than the number of PCPs the system has in real will introduce a lot of overhead in the VMM and incur cache and CPU contentions, which finally degrades the performance of the whole platform.

3.5 Appendix
Figure 3.10: Statistics of [1 Asterisk VM with 1 VCPU]
Figure 3.11: Statistics of 1 Asterisk VM with 2 VCPUs
Figure 3.12: Statistics of [2 VMs on CPU competition, 1 VCPU per VM]
Figure 3.13: Statistics of [2 VMs on CPU competition, 2 VCPUs per VM]
Figure 3.14: Statistics of [2 VMs on Network I/O competition, 1 VCPU per VM]
Figure 3.15: Statistics of [2 VMs on Disk I/O competition, 1 VCPU per VM]
Figure 3.16: Statistics of [2 Identical Asterisk VMs, 1 VCPU per VM]
Chapter 4

Performance Characterization
Framework of Virtual Appliances

4.1 Introduction

Based on our experience developing a performance framework for the AsteriskNOW virtual appliance discussed in the last chapter, we have designed VAMark, a framework for performance characterization of virtual appliances. As part of this new framework, we propose a metric for reporting on the resource needs of a virtual appliance. The normalized performance score is used to characterize the behavior of the virtual appliance. A standardized competitor VM, VMmeter is developed as part of this new framework. VMmark can be used for characterizing performance in any virtualized environment (VMM-independent).

To illustrate the design and utility of this new framework, we have implemented the framework on top of the Xen virtual machine based on an Intel dual-processor dual-core based machine. We evaluate the performance of three different workloads
running in a virtual appliance environment:

1. SPECint 2006 benchmarks [30]


3. Apache web server [14]

In this chapter, we present the VMmark framework and our methodology used to characterize performance. We then describe the VMmeter design, the test workloads, and system configurations. The characterization results and analysis of the virtual appliances are presented and discussed. We also provide profiling data to locate the hottest part in the virtual appliance.

4.2 VAmark Framework

The structure of our virtual appliance performance characterization framework VMmark is shown in Figure 4.1. The framework consists of three parts:

1. A workload generator for the virtual appliance

2. The platform under test (including the virtual appliance, VMmeter and VMM)

3. A profiling tool that collects and analyzes VMM statistics

A client-based VA workload generator on a separate physical machine drives the target VA application with load. This environment mimics how a VA would be
deployed in practice where remote systems would be accessing services from the VA server. The application-level performance data (i.e., performance scores) are gathered from the VA workload generator. The VMM-level profiler probes the VMM to provide VM-level information, including VM status, virtual/physical CPU utilization, VMM scheduling statistics, I/O speed, etc..

To fully characterize the sharing present on the system, we have created a separate VM called VMmeter, which runs a range of configurable benchmarking programs on a thin Linux operating system. VMmeter allows us to generate intensive workloads

Figure 4.1: Overview of our VAmark framework.
that can compete for selected hardware resources. (We will describe more details in the next section.

To quantify the performance of the virtual appliance under test, we use the performance degradation experienced when running VMmeter versus an unloaded baseline performance of the VA with VMmeter idle; we report this as a normalized performance score. Since we can stress up to four different hardware resources when running VMmeter (CPUs, memory, disk, and network), we generate four different performance scores to provide a more complete characterization of the virtual appliance. The performance degradation of VMmeter represents the performance interference incurred by the virtual appliance.

Furthermore, in order to profile and analyze the whole platform, a system-wide profiler is needed that is capable of profiling all running guest VMs and the underlying VMM. But one key requirement of a system-level profiler is that it cannot add any significant overhead to the system under test. To satisfy this requirement and still produce data-rich results, we obtain execution profiles using the hardware performance counters of the CPU [22]. Hardware counters have been used in a number of previous studies [6, 23]. Using our approach, a large number of events can be profiled across the following execution domains:

- hardware and software interrupt handlers,
- the Linux kernel,
• kernel modules,

• shared libraries, and

• executable binaries.

By collecting and analyzing system-level statistics, we can quickly and accurately locate the source of any performance overhead. We analyze both the VMM and the virtual appliance by extracting detailed runtime information and architectural statistics. Our framework provides the ability to drill down to individual functions in any execution domain, while capturing the frequency of hardware-related events.

4.3 VMmeter Design

VMmeter is a separate VM that runs next to the virtual appliance to be tested. VMmeter contains a thin Linux operating system that has small guest operating system overhead. Inside the guest system, VMmeter runs a benchmark suite that allows testers to generate intensive workloads that compete for selected hardware resources. The VMmeter VM has access to all available hardware resources (i.e., all of the CPU cores, disk adapters and network interfaces) on the virtualized hardware.

We measure the execution time of these competing workloads give a specific level of load within VMmeter. VMmeter creates a simulated environment where the virtual appliance under test shares the hardware resources with other VMs and performance interference incurred in this environment. Currently, we have created exercisers that
stress CPU, memory, disk I/O and network I/O resources. This benchmark suite is written all in the C language. Table 4.1 provides additional details of these four benchmarking elements of VMmeter.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Test</td>
<td>Calculating prime numbers using the Euclid algorithm; capable of generating multi-threaded execution.</td>
</tr>
<tr>
<td>Memory Test</td>
<td>Generates a stream of sequential memory reads and writes to stress the memory hierarchy.</td>
</tr>
<tr>
<td>Disk I/O Test</td>
<td>Generates a stream of random reads and writes on multiple large files.</td>
</tr>
<tr>
<td>Network I/O Test</td>
<td>Testing network throughput by transferring data from a client to the server over TCP/IP.</td>
</tr>
</tbody>
</table>

Table 4.1: The current benchmarking elements present in VMmeter.

4.4 Experimental Setup

For the following experiments, we use an Intel dual-processor dual-core Xeon 5140 CPU with Intel VT-x [28], running at 2.33GHz with 4MB L2 Cache, 4GB main memory, SCSI disk and gigabit ethernet as the target machine. We have totally 4 physical CPU cores (PCPs).

We have implemented this framework based on the Xen virtual machine. We use Xen 3.2, the 64-bit version with both Domain-0 and guest domains running 64-bit openSuSE Linux 11.0. In order to optimize performance, we utilize para-virtualized guest VMs. For the hypervisor scheduler, we use the Credit Scheduler [10] which provides fair CPU scheduling in an SMP environment.
Xen VMM statistics (e.g., CPU utilization, disk I/O counts, network throughput) were collected using xentop. We also used xenoprof \cite{23} to capture performance counters on the CPU. This allows us to profile both the host VMM and guest VMs, and provides us with a better picture in terms of host versus guest execution interference.

VMmeter is configured as a VM with 1GB of main memory that has access to all available PCPUs. The target VA is setup with 2GB of main memory and two VCPUs. We run the SPECint 2006 benchmarks, MySQL database server and Apache web server. We report on run time performance of three specific SPECint 2006 benchmarks in this thesis, though have generated results all of the benchmarks in the suite.

- **SPECint 2006 benchmarks** - the SPEC benchmarks are compiled for a 64-bit Linux platform. The default reference input from the SPEC benchmark suite is used for test, and are run to completion.

- **MySQL database server** - we use the MySQL build 5.0.51, which comes packaged as a database server. To drive database testing, we use the Wisconsin Benchmark \cite{8} provided by MySQL AB. The workload exercises different types of queries, providing a rich database behavior.

- **Apache web server** - we use the Apache version 2.2.8 as the web server to serve static webpages. We use httperf \cite{25} to serve as a client to drive Apache.
Httpperf generates HTTP workloads and measures the performance of the web server in terms of the rate of pages served.

We use another Intel dual-core Xeon 2.4GHz machine to run httpperf that generates webpage requests to the Apache web server and traffic for the network I/O benchmark in VMmeter VM. The systems are connected via a gigabit network switch.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Intel Xeon 5140 - 4 PCPUs, 4GB Memory, SCSI Disk, Gigabit NIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA</td>
<td>2 VCPUs, 2GB Memory</td>
</tr>
<tr>
<td></td>
<td>Workload: SPECint 2006, MySQL Database, Apache Web Server</td>
</tr>
<tr>
<td>VMmeter</td>
<td>4 VCPUs, 1GB Memory</td>
</tr>
<tr>
<td></td>
<td>Stress Tests: CPU, Memory, Disk I/O, Network I/O</td>
</tr>
</tbody>
</table>

Table 4.2: Summary of our experimental VA platform.

During all of our experiments, the virtual appliance VM and the VMmeter VM share the same physical SCSI disk and the gigabit ethernet card. For the PCPUs we consider two different virtual machine settings:

1. VCPUs of the virtual appliance are floating on all PCPUs (no CPU affinity is the default setting)

2. VCPUs of the virtual appliance are pinned to specific PCPUs (i.e., CPU affinity)

We configure VMmeter to run both on the the same and different PCPUs as the virtual appliance VM to evaluate the impact of this configuration choice. For all the experiments, Domain-0 is allowed to run on any PCPU.
4.5 Results and Analysis

4.5.1 Performance Characterization

Figure 4.2 shows the resulting performance scores of running the virtual appliances with VMmeter, while not specifying any CPU affinity. This particular environment is unconstrained in terms of CPU resources, since we have as many physical CPU resources as we have VCPUs. By not specifying affinity, we allow the two VMs to utilize the available CPU time across the 4 cores. All three of our applications (we treat the 3 SPEC benchmarks as one workload) are able to achieve greater than 99% of their baseline performance, when competing with VMmeter. The results also indicate the capability of the VM scheduler in Xen is able to effectively balance multiple workloads on such a SMP system. The performance is measured from the perspective of only the virtual appliance.

We found that the VMmeter Network I/O Test impacted the SPECint2006
benchmarks slightly (around 5% degradation). The network-intensive workload in VMmeter transfers data over a gigabit network. Since we do not restrict the network throughput of VMmeter, the network test reaches an average throughput of 113 MBps (see Table 4.3). Domain-0 is frequently involved in handling network I/O processing, using 32% of the PCPU resources. The PCPU usage of Domain-0 is normally 1-2% for non-I/O intensive workloads. The SPEC benchmarks interact with Domain-0 insignificantly, which explains the small about of impact.

<table>
<thead>
<tr>
<th>Domain</th>
<th>CPU(%)</th>
<th>NET TX (KB/s)</th>
<th>NET RX (KB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-0</td>
<td>32.13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VA</td>
<td>99.71</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>VMmeter</td>
<td>20.93</td>
<td>115857.39</td>
<td>2625.69</td>
</tr>
</tbody>
</table>

Table 4.3: Resources utilization of running gcc vs. VMmeter network I/O test.

Next we evaluate how VMmeter is impacted when running in this same environment. From Figure 4.2(b) we can see that MySQL is competing with the VMmeter Disk I/O load. The VMmeter Disk I/O test experiences 6% performance degradation. This contention is due to the frequent disk accesses present in the MySQL database server workload. Table 4.4 shows statistics of running the MySQL database VA in terms of disk operations per second. The MySQL database VA can be categorized as both CPU and disk I/O Intensive. We present a more detailed analysis of this workload in the next section.
<table>
<thead>
<tr>
<th>Domain</th>
<th>CPU(%)</th>
<th>Disk Operations/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-0</td>
<td>3.63</td>
<td>-</td>
</tr>
<tr>
<td>VA</td>
<td>95.4</td>
<td>221.55</td>
</tr>
<tr>
<td>VMmeter</td>
<td>0.81</td>
<td>78.15</td>
</tr>
</tbody>
</table>

Table 4.4: Resource utilization of MySQL vs. VMmeter disk I/O test.

4.5.2 Performance Impact of Sharing PCPUs

To evaluate the impact of sharing physical CPUs (which would occur on any platform where there are more VCPUs than PCPUs), we configured the system identical to the previous set of experiments, with the exception of pinning VCPUs of the virtual appliance to two specific PCPUs. Two threads were created by VMmeter that run on the same two PCPUs as the virtual appliance under test. Since the two VMs are competing for the same hardware resources, most of the applications suffered significant performance degradation (see Figure 4.3).

As shown in Figure 4.3(a), the performance scores of the Apache web server VA remain at 100% of the baseline performance. We configured the Apache workload driver to issue webpage requests at the maximum rate that the Apache server is capable of handling. For this particular workload the performance scores are measured by the rate at which webpages can be served. As shown above, this rate from the VA application perspective is not impacted by VMmeter.

Figure 4.3(a) shows that CPU-intensive appliances experience a 45% performance
degradation while running the VMmeter CPU Test. At the same time, the VMmeter CPU Test sees a 25% execution slowdown when testing the appliances (see Figure 4.3(b)). This variation in slowdown makes sense since VMmeter fully utilizes the two VCPUs by creating two threads, though the SPECint2006 benchmarks are running in a single thread. In short, CPU-bound virtual appliances sharing PCPUs will experience significant overhead and degradation in performance. Table 4.5 lists the statistics of the gcc SPEC benchmark run against the VMmeter CPU test. The performance degradation of gcc is almost linear with respect to the decrease in CPU utilization in the VMmeter CPU test.

From Figure 4.3(a), we find that in the VMmeter network I/O test, both the SPEC CPU benchmark suite and the MySQL server experience a 10-20% execution slowdown. This is due to the large amount of network I/O activity that causes Domain-0 to remain tied up handling both I/O processing and VM switching. We
analyze the details of this scenario in the next section (Case Study 2) to better understand how VMmeter affects the virtual appliance in the network I/O test.

<table>
<thead>
<tr>
<th>Application</th>
<th>Restrained CPU</th>
<th>VM Statistics</th>
<th>Performance Degradation</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Domain-0: 1.32</td>
<td>VA: 45%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VA: 102.76</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VMmeter: 196.57</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>Domain-0: 0.79</td>
<td>VMmeter: 25%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VA: 50.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VMmeter: 147.97</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5: VM-level Statistics of gcc vs. the VMmeter CPU test.

4.6 Profiling and Analysis

In order to better understand and pinpoint performance bottlenecks, low-level system profiling is needed. We concentrate on two aspects when profiling the whole system:

1. the distribution of hardware events occurring across different applications/functions, and

2. quantitatively measuring differences between running a virtual appliance alone versus with VMmeter.

In this thesis we analyze three hardware events:

1. CPU_CLK_UNHALT: the number of clock cycles spent in the unhalt state - this number roughly estimates the CPU busy time spent while executing a binary.
2. \textit{L2.CACHE.MISS}: the number of L2 cache misses - this number captures the number of memory references that miss in the L2 cache and access main memory.

3. \textit{ITLB.MISS}: the number of ITLB misses - this number provides a rough estimate of the frequency of context switches and VM switches.

We could consider additional events, though this set provides us with ample information to illustrate the value of our approach.

In the following case studies, we consider affinity-based PCPU configurations. For the VA under test, we focus on the hottest portions of the execution. The overhead experienced by the VA is quite small when compared to the baseline runs. For the scenario where VMmeter competes with the VA, we want to see what the impact is in terms of hardware events in Domain-0. We evaluate this next.

\textbf{4.6.1 Case Study 1: MySQL vs. VMmeter Disk I/O Test}

Table 4.6(a) shows the percentage of CPU busytime in the virtual appliance for program functions that are CPU-dominant. Binaries \texttt{mysqld} and \texttt{perl} are the MySQL daemon and the testing client written in Perl, respectively. The \texttt{toggle.guest.mode} binary also consumes some CPU cycles to toggle between kernel and user mode and performs page tables swapping, accordingly. Table 4.6(b) shows an increase in CPU busytime in Domain-0. When the VMmeter disk I/O test is run with the virtual appliance, since both VMs are disk I/O intensive, we observe significant increases in
disk-related functions (e.g., disk driver \texttt{aic94xx} and IRQ services). The last function, \textit{csched\_schedule} is the Xen scheduler that switches between guest VMs; we observe a 200\% increase in CPU cycles.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysqld</td>
<td>36.10</td>
</tr>
<tr>
<td>perl</td>
<td>18.03</td>
</tr>
<tr>
<td>toggle_guest_mode</td>
<td>1.83</td>
</tr>
<tr>
<td>libpthread</td>
<td>1.00</td>
</tr>
</tbody>
</table>

(a) Virtual Appliance - CPU Usage

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Sample Increase(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop device</td>
<td>133</td>
</tr>
<tr>
<td>aic94xx</td>
<td>379</td>
</tr>
<tr>
<td>_spin_lock_irqsave</td>
<td>206</td>
</tr>
<tr>
<td>csched_schedule</td>
<td>192</td>
</tr>
</tbody>
</table>

(b) Domain-0

Table 4.6: Summary of CPU busytime of running MySQL.

Table 4.7(a) shows that the top functions experience a significant number of L2 cache misses. The MySQL daemon and Perl are responsible for most of these misses. String copy and memory copy operations also generate a significant number of L2 cache misses. In Domain-0 (shown in Table 4.7(b)) we see an increase of 20\% in the number of L2 cache misses. Most of these missed are related to managing disk operations. The Xen scheduler experiences additional L2 cache misses due to the memory collisions generated when multiple VM contexts share the same memory hierarchy concurrently.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysqld</td>
<td>41.62</td>
</tr>
<tr>
<td>copy_user_generic_string</td>
<td>6.63</td>
</tr>
<tr>
<td>perl</td>
<td>5.13</td>
</tr>
<tr>
<td>_copy_user_nocache</td>
<td>3.76</td>
</tr>
</tbody>
</table>

(a) Virtual Appliance - L2 Cache Miss

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Sample Increase(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop</td>
<td>17</td>
</tr>
<tr>
<td>blk_rq_map_sg</td>
<td>22</td>
</tr>
<tr>
<td>csched_schedule</td>
<td>18</td>
</tr>
<tr>
<td>free_block</td>
<td>15</td>
</tr>
</tbody>
</table>

(b) Domain-0

Table 4.7: Summary of L2 cache misses for MySQL.
Next, we examine the distribution of ITLB misses. ITLB misses indirectly tell us which functions cause context/VM switches in Xen. Table 4.8(a) shows ITLB misses across different functions. Since our VA is para-virtualized, hypercalls are generated whenever an event occurs that requires the Xen hypervisor to handle the privileged operations. For instance, `update_cr3` is the hypercall to change the page table base register and `hypercalle_page` is actually a code page, which contains 32 hypercall entries. ITLB misses occur when a hypercall is made. The increase in the number of ITLB misses in Domain-0 is low (about 10%), as shown in Table 4.8(b).

<table>
<thead>
<tr>
<th>Function Name</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysqlld</td>
<td>16.32</td>
</tr>
<tr>
<td>perl</td>
<td>8.10</td>
</tr>
<tr>
<td>toggle_guest_mode</td>
<td>7.72</td>
</tr>
<tr>
<td>update_cr3</td>
<td>4.34</td>
</tr>
<tr>
<td>hypercall_page</td>
<td>2.76</td>
</tr>
</tbody>
</table>

Table 4.8: Summary of ITLB misses of running MySQL.

### 4.6.2 Case Study 2: gcc vs. VMmeter Network I/O Test

Table 4.9(a) highlights the *hottest* functions in both the SPECint 2006 virtual appliance and Domain-0 in terms of CPU usage when running the VMmeter network I/O test. The CPU usage in our virtualized appliance is almost exclusively used by the `gcc` functions and `glibc`. This is not surprising since gcc is primarily CPU-bound. In the baseline benchmark run, Domain-0 is only introducing 1-2% CPU overhead.

To further investigate how network I/O impacts Domain-0, we analyzed the CPU
utilization statistics in Domain-0. CPU usage by the network driver e1000e is increased from 0 to 11%. This is the largest contributor to CPU usage, followed by the functions bridge and netbk. The bridge is the virtual-to-physical interface and netbk is the network interface front-end in Domain-0. Results are shown in Figure 2.2; these functions are responsible for the overhead we observed in the performance scores. The Xen scheduler is frequently switching VMs. We also found that both L2 cache misses and ITLB misses are nearly proportional to the CPU utilization of both Domain-0 and the gcc VA.

<table>
<thead>
<tr>
<th>Application</th>
<th>Function</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>libc-2.8.so</td>
<td>-</td>
<td>14.86</td>
</tr>
<tr>
<td>gcc_base</td>
<td>reg_is_remote_constant_p</td>
<td>9.62</td>
</tr>
<tr>
<td>gcc_base</td>
<td>clear_table</td>
<td>4.35</td>
</tr>
<tr>
<td>gcc_base</td>
<td>bitmap_operation</td>
<td>3.83</td>
</tr>
<tr>
<td>gcc_base</td>
<td>single_set_2</td>
<td>2.79</td>
</tr>
</tbody>
</table>

(a) Virtual Appliance - CPU Usage

<table>
<thead>
<tr>
<th>Function Name</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1000e</td>
<td>11.22</td>
</tr>
<tr>
<td>bridge</td>
<td>6.42</td>
</tr>
<tr>
<td>netbk</td>
<td>5.07</td>
</tr>
<tr>
<td>csched_schedule</td>
<td>2.94</td>
</tr>
<tr>
<td>aic94xx</td>
<td>2.43</td>
</tr>
</tbody>
</table>

(b) Domain-0

Table 4.9: Summary of CPU busytime when running gcc.

### 4.7 Summary of VAmark

In this chapter, we presented VAmark, a performance characterization framework for virtual appliances. To illustrate the utility of our framework, we develop a testing environment based on Xen virtual machine and characterized the performance behavior of selected benchmarks from the SPECint 2006 suite, MySQL database server and Apache web server applications. We described VMmeter, a tool that can create
load on our virtual appliance. We investigated performance degradation of both the VAs and the VMeter load. We further presented a detailed profiling analysis, which allowed us to dive into the reasons for contention in our VA environment. We believe that our VAmark framework can be useful to guide the design and deployment of virtual appliances.
Chapter 5

Conclusion and Future Work

5.1 Contributions of this Thesis

Virtualization is becoming commonly used in large datacenters and computing infrastructures. While costs can be reduced by consolidating servers on a virtualized platform, current virtualization technologies provide inadequate performance isolation. This issue is particularly true on virtualization appliance servers, where multiple virtual appliances share common hardware. We need to understand the performance characteristics of the virtual appliances in order to predict how they will scale. This thesis has both investigated this issue, and proposed a new methodology for quantitatively evaluating this VA performance.

Our experiments have shown that the performance of a virtual appliance suffers considerable degradation when sharing hardware resources with other virtual machines, though when VAs are mixed that stress unique hardware resources, VAs can scale more gracefully.
Our work has allowed us to develop a better understanding of how the virtual appliances behave and our metrics can help guide an IT administrator or system integrator when making decisions about provisioning the available server resources. Using our VAmark, we are able to identify the source of any performance overhead and quantify performance interference. We envision that future users of our framework analyze the performance and develop a set of best practices to guide the design and deployment of virtual appliances.

The key contributions of this thesis include:

- we have presented a comprehensive performance evaluation study of AsteriskNOW virtual appliance, which has helped to guide the design of our VAmeter framework,
- we have proposed the concept of a normalized performance score to quantify the performance interference when running multiple virtual appliances,
- we have developed VMmeter as a standard competitor virtual machine. It runs different benchmarking tests to generate workloads and create resources competition, and
- we have designed the VAmark framework and implemented it; we evaluated and analyzed the performance of three different workloads running in a virtual appliance environment.
5.2 Future Work

Our work on virtual appliance benchmarking and performance analysis is one of the first attempts at quantifying performance in this dynamic environment. Future work on this topic could be pursued in a number of directions. One possible direction is to improve upon the characterization methodology when multiple copies of the same virtual appliance or a mix of consolidation workloads are running concurrently. Presently this is a labor-intensive activity. We need to assess how best to characterize the behavior of each workload and quantify the associated performance interference. The goal will be to develop an analytical modeling environment that only works with workload characteristics to predict performance and scalability.

Our future work will also consider a wider range of VM configurations. We need to consider the impact of priorities and shares that will also impact performance and scalability.

Our VAmark is a VMM-independent characterization framework. We have implemented our framework on both Citrix’s Xen and VMware ESX Server. We plan to consider additional VMMs including Microsoft’s virtualization offerings such as Hyper-V.

We also need to examine virtualization overhead at a finer granularity. To determine this, we need to measure the performance of a software appliance on a native
platform (without running VMM/hypervisor) and compare the performance on a virtual machine. The events such as context switches, interrupts and page faults in two different environments will need to be measured. The cost of these events may vary under virtualization. It would be helpful if processor-level information such as processor traces or debugging feature is available for use to analyze the virtualization process.
Bibliography


