TOWARDS AUTONOMIC VIRTUAL MACHINE MANAGEMENT

By

SIDDHARTH WAGH

A thesis submitted to the

Graduate School – New Brunswick

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Master of Science

Graduate program in Electrical and Computer Engineering

Written under the direction of

Dr. Manish Parashar

And approved by

____

____

____

____

New Brunswick, New Jersey

OCTOBER, 2010

ABSTRACT OF THE THESIS

TOWARDS AUTONOMIC VIRTUAL MACHINE MANAGEMENT

By SIDDHARTH WAGH

Thesis Director:
Dr. Manish Parashar

Virtual machine technologies are gaining wide acceptance in today’s era due to invaluable services in system management, server consolidation, and secure resource containment along with providing requisite application execution environment. Every virtual machine platform reduces dependence on hardware by fully or partially abstracting operating systems enabling flexible control of manipulation or migration of guest machines by manual system administration or reactive/proactive approaches to management. This dissertation focuses on resolving the resource reservation problem to help define a mathematical model and study interference within multiple virtual machines while trying to achieve load balancing and improve performance efficiency.

Our goal is three-pronged. Firstly, we aim to understand the underlying support available for virtual machine migration and pursue new technologies or abstractions to improve efficiency and speed of the data transfer. Secondly, we carefully evaluate all the resources used by VMs for proper functioning and study the synchronization and multiplexing processes underneath which delineate when and where to migrate a virtual machine. Finally, we attempt to deduce the action to perform on running VMs (manipulation or resource configuration) so as to resolve the issue at hand. To achieve these goals, we follow a step-by-step procedure limiting the number of variable parameters and analyze the outcome of focal experiments.

The results show that, using RDMA (Remote Direct Memory Access) to perform virtual machine migration can be used only in scenarios where the underlying hardware offers support for such transactions (eg. InfiniBand architecture) and such an abstraction over TCP/IP does not ameliorate efficiency of VM transfers. Further, a utility based function designed to analyze environment and application metrics and project an area of good/bad states on a map would require a plethora of parameters increasing its complexity. Considering VM re-distribution, one can predict the ideal number and time of migration of guest virtual machines on any configuration by gathering statistics from parallel migration for graphical analysis. Parallel VM migration gives us shorter average transfer time and higher latencies per VM. Pinning of virtual CPUs to VMs improves the performance efficiency of applications compared to sharing of CPUs.

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my thesis advisor Professor Manish Parashar for his able guidance, motivation, sound advice and financial support during the two years of my research work at TASSL (The Applied Software Systems Laboratory). I am grateful to my colleagues at TASSL, especially Dr. Ivan Rodero, Ciprian Docan and Jim Housell for their unconditional help with system issues and providing valuable insights in my work. I salute all the staff & students at Center for Autonomic Computing and Department of Electrical & Computer Engineering for timely assistance during various stages of my Masters. I am forever indebted to my family, especially my parents Asha & Vijay Wagh, for the mantra "Prayers, Efforts and Perseverence" and their whole-hearted financial and emotional support for my studies abroad. I owe an extravagant party to my roommates Indraneel, Jay and Prashant; pseudo roommates Rajat, Sahana & Milind and my dear friends at Rutgers for being my family away from home. Last but not the least, I wish to thank Pranav, Pooja, Amit and all my wonderful friends and cousins in India for just being a phone call or a chat buzz away when I needed them.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1 INTRODUCTION

Today’s data centers are becoming computationally complex and faster due to advances in x86 virtualization technologies supported by speedier processors. Server consolidation and efficient resource provisioning are two primary areas of focus in the field of virtual machine management. Current VM technologies surpass many limitations such as sub-optimal resource utilization, power wastage on idle physical machines, unbalanced workloads etc. by allowing a single physical server to spawn multiple virtual machine “sandboxes”, each maintaining a strong, secure and isolated execution environment for its applications. These containers can be customized at a fine granular level so as to meet the needs of the complete software and hardware platform required by applications, control access to resources and support timely deployments.

The manipulation of these virtual machines for effective resource configuration and lifecycle management can be carried out by different interfaces provided by underlying virtualization platforms. This concept of scheduling VMs on hardware is analogous to processes running typically on operating systems, allowing time and space sharing of resources through careful multiplexing handled by a central privileged monitor. Each VM container may run a specific OS distribution, provided it supports paravirtualization and has a modified kernel module for valid interaction with hypervisor.

1.1 Motivation and Problem Description

Server consolidation, optimum resource utilization and disaster recovery are the prime reasons for adoption of virtualization in a data center environment. A large influx of VM technologies in the market signifies the dire need for backend data centers to utilize their limited physical resources in the most optimal way to handle enormous quantity of data. Though every technology offers specialized features depending on customer needs, manual intervention by system administrators is essential for maintenance and re-configuration. The field of research focusing on automating VM management is growing actively issuing multiple approaches for system learning and task automation. We present our work in this light contributing partly to the reinforcement learning methodology by a comprehensive evaluation of VM behavior. Our choice of Xen to explore the functionality offered by virtualization platform is primarily because of its open source nature and a strong community of developers about it. Though Xen provides an array of features to create, suspend or migrate VMs on the run, it leaves the area of automating the process of VM management for further research.

Our work focuses on VM-based resource reservation (namely CPU, memory, network, disk resources) and automation of VM operations to support load balancing and server consolidation. The fundamental aim is to enable the physical server to take proactive decisions regarding creation, migration or shutting down of virtual machines based on an inbuilt model governed by environmental specifications of guest virtual machines and SLAs. Time sharing of resources at a coarse granularity is of prime concern resulting in optimal resource utilization for different workloads at different times of the day. This involves timely manipulation of all hosted VMs in a cluster triggered by crossing over of pre-decided thresholds or signaling events.

1.2 Scope of Research

Virtualization technology has matured over the last decade spawning a variety of projects and ideas for effective VM management. The following studies give us a broad outlook of research in the area of efficient energy utilization and power management. (D. A. Menasce 2006) propose an autonomic controller portraying its dynamic use in allocating CPUs in virtualized environments with varying workload levels by optimizing a global utility function. (A. Bertl 2009) express the need for energy efficiency in information systems and review various methods to achieve it. They highlight certain key research challenges in cloud computing environments. For server systems, (J. S. Chase 2001) discuss how to reduce power consumption in data centers by turning servers on and off on demand.

Looking at the current trends in virtualization, one preliminary article is by (M. Rosenblum 2005) which highlights the future of virtual machine monitors. A significant proposal by (W. Voorsluys 2009) on performance evaluation of live migration of VMs on running applications goes parallel to our work, and so does the paper by (M. Zhao 2007) which is an experimental study. Many reactive approaches have been proposed over the years; for instance, VMPlants (I. Krsul 2004) which talks about managing grid environments in grid computing or VManage (S. Kumar 2009) for loosely coupled platforms. Very few researchers have ventured into buttressing the underlying infrastructure for expediting migration. (W. Huang 2007) envisaged the usage of InfiniBand architecture for high performance virtual machine migration. A more detailed analysis is provided in Section 2.3.

Our goal is to perform a comprehensive analysis of the reconfiguration issues in a virtual environment and build a mathematical model encompassing various parameters highlighted in the same. Various challenges are encountered in the process owing to the plethora of control parameters responsible for the operations of virtual machines. The underlying hardware and server environment greatly affect the results obtained by running exhaustive tests. Deducing task perceived utility from different running applications produces different scenarios for similar server configurations.

1.3 Contribution

This research is an exploratory approach towards autonomic virtual machine management experimenting with different advances towards the task of resource re-configuration and enhancing performance efficiency. The key contribution of the matter is a comprehensive analysis of CPU scheduling policies on a Xen platform on a credit scheduler module suggesting ways to manipulate virtual machines based on active monitoring of CPU utilization on a fragmented level. Further, our venture is an exhaustive study of parallelization of system operations and its effect on the application or its environment. Compared to related work, our analysis considers more varied parameters and seeks to leverage new technologies to aid virtual machine management in a controlled environment.

1.4 Thesis Organization

This document is organized into six chapters as follows:

Chapter 1 introduces the concept of virtualization and its mechanics. It presents the motivation behind following this research track and describes the problem at hand. It provides a summary of existing work, current trends in VM technology and the set of challenges facing the researchers. Finally, it highlights our contribution to the field of virtualization and performance isolation in data centers.

Chapter 2 lays the foundations for our research by explaining the framework behind our studies and showcasing different platforms and tools. It begins by presenting the basics of virtualization, its benefits and its different types classified either by the resource being virtualized or the proximity to hardware layer. It delves into the Xen platform, its hypervisor, monitoring tools and domain management. The final section presents related work in detail.

Chapter 3 and 4 highlight our different approaches to resource re-configuration and multiplexing, along with improving underlying migration support and applying utility based logic to VM management. It contains rigorous analysis using different benchmarks on our existing physical machines. The methodology behind each approach is presented in algorithmic format to outline a general experimental procedure.

Chapter 5 contains the results derived from the analysis done in previous chapters. It uses various graphs and tables as aid to present significant findings. Chapter 6 encompasses the conclusions deduced from the results, summarizes the exercise and lists the tasks for future work.

2 BACKGROUND AND RELATED WORK

Virtualization, in the broad sense, involves differentiating a service request by an application from the physical delivery of the service in hardware (VMWare 2007). A software abstraction layer is added between the hardware and the operating system running on the system. The resultant layer performs the following functions: (1) it allows multiple operating systems to run simultaneously on a single physical machine; (2) it dynamically partitions and isolates available physical resources such as the memory, CPU(s), disks and I/O devices. Such a powerful technology has immediate benefits such as enabling server consolidation, providing optimum resource utilization, simplifying software development and testing, and enhancing agility in data centers.

The limitations owing to dependence on hardware alone have been reduced to a great extent, thanks to fully abstracted operating systems and applications which enable a plethora of virtualization features after their encapsulation into portable virtual machines. Let us consider the industry-standard x86 architecture for instance. X86 virtualization employs either hosted or hypervisor architecture. If the virtualization layer is installed on bare-metal (directly on an x86-based platform), it is called a hypervisor architecture; else if the layer runs as an application on the top of a host operating system, it is called a hosted architecture. Having direct access to all physical resources rather than through an OS makes a hypervisor more efficient, scalable and robust than hosted architecture. Servers can run on virtual infrastructure 24x7 all year round, with no downtime required for maintenance and over fault tolerant configurations. Examples of a hosted architecture include VMWare Player, ACE, and Workstation etc. while hypervisor architecture exists on ESX Server and open-source platform Xen (used in this research).

2.1 Virtualization Basics

To delve deeper into x86 virtualization, we hereby analyze its component parts. Figure 1 shows the basic structure of a virtualized system. The base virtualization layer is the software which hosts and manages different virtual machines on virtual machine monitors (VMMs). It is the hypervisor running directly on a clean slate of hardware. Its functionality differs for various architectures and implementations. Every VMM running on the hypervisor has its own virtual machine abstraction responsible for running a guest OS. Its main duties are to partition and share the different resources like CPU, memory, disk and I/O devices so that the entire system is shared by effective multiplexing between different VMs.

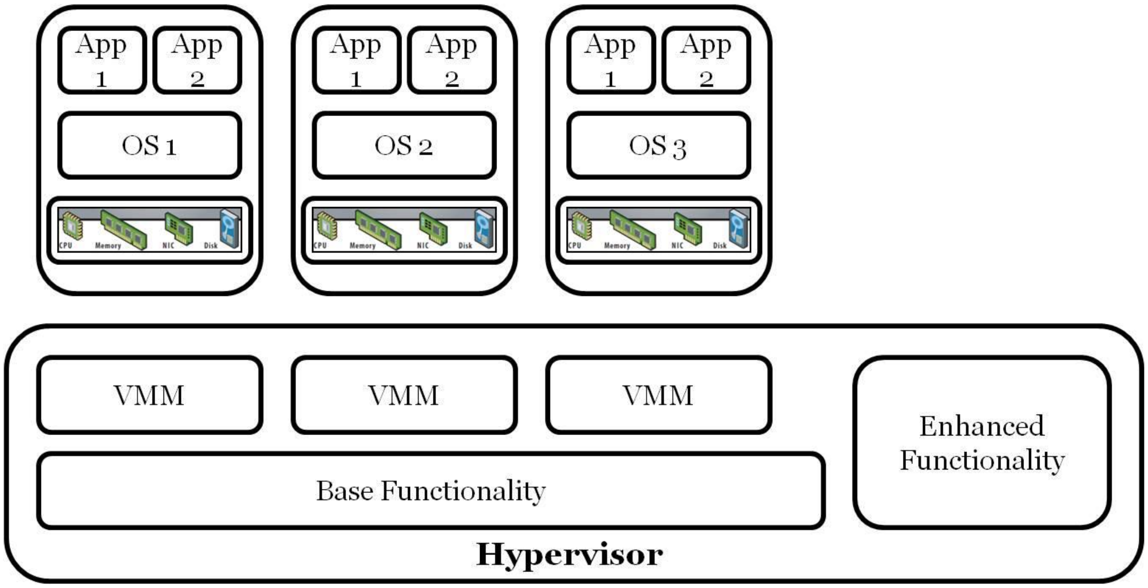


Figure 1: Basic Structure of a Virtualized System

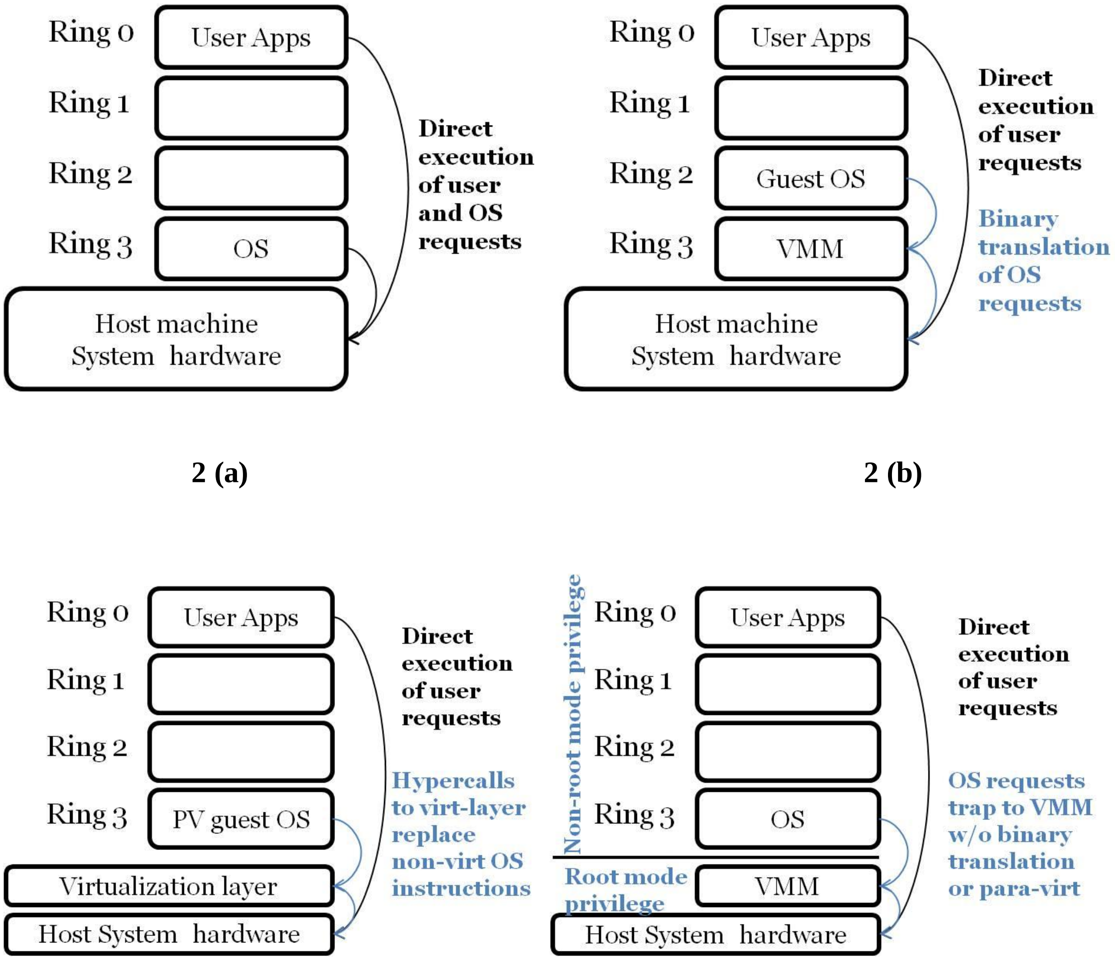
2.1.1 Types of Virtualization

Virtualizing the x86 architecture involves working directly on bare-metal hardware, thus necessitating the system to have full control of the physical resources. Figure 2 shows the four different privilege levels inherent to x86, known as Ring 0, 1, 2 and 3 denoting access levels to the computer hardware. Ring 0 is primarily used by user applications, Ring 3 by operating system to execute its privileged instructions directly on system resources and Ring 1 & 2 are rarely used. To incorporate virtualization layer in this scenario requires designers to place it under the OS (more privileged than Ring 0) to create and manage guest OS/VMs so as to share resources. To add to the complexity, some sensitive privileged instructions are hard-coded to run in Ring 0 only and to trap and translate such instructions has been one of the many challenges in virtualization technology.

Based on different resources, virtualization can be classified into following categories: (1) CPU Virtualization, (2) Memory Virtualization and (3) Device and I/O Virtualization. CPU virtualization can be further divided into three types, depending on mechanisms used to handle privileged instructions and flexibility to be achieved; namely (1) Full virtualization using Binary Translation, (2) Para-virtualization or OS-assisted virtualization and (3) Hardware assisted virtualization.

2.1.1.1 CPU Virtualization

The three techniques used to achieve CPU virtualization communicate via the different privilege rings in a unique manner which has been presented for comparison in Figure 2 (b), (c), (d) against 2 (a) showing a non-virtualized system.


Ring 0
User Apps
Ring 1
Ring 2
Ring 3
OS
Host machine System hardware
Direct execution of user and OS requests
Ring 0
User Apps
Ring 1
Ring 2
Guest OS
Ring 3
VMM
Host machine System hardware
Direct execution of user requests
Binary translation of OS requests
Ring 0
User Apps
Ring 1
Ring 2
Ring 3
PV guest OS
Virtualization layer
Host System hardware
Direct execution of user requests
Hypercalls to virt-layer replace non-virt OS instructions
Non-root mode privilege
Ring 0
User Apps
Ring 1
Ring 2
Ring 3
OS
Root mode privilege
VMM
Host System hardware
Direct execution of user requests
OS requests trap to VMM w/o binary translation or para-virt

2 (a)

2 (b)

2 (c)

2 (d)

Figure 2: (a)Non-virtualized system, (b)Full Virtualization using Binary Translation, (c)Paravirtualization, (d) Hardware Assist Virtualization

2.1.1.1.1 Full virtualization using Binary Translation

This technique involves a combination of binary translation (translation of kernel code to replace non-virtualizable code with new modified set of instructions to have desired effect) and direct execution (direct user level code execution on processor) so that every VMM provides a set of virtual BiOS, devices and memory management modules. Since the hypervisor translates all OS instructions on the fly leaving user code to run unmodified, the OS is completely decoupled from the underlying hardware. Some benefits achieved by this technique include complete isolation, high security, portability and efficient migration of VMs. Eg. VMWare products, MS Virtual Server.

2.1.1.1.2 Paravirtualization (OS-Assisted Virtualization)

Paravirtualization requires modifications to the guest OS kernel, particularly replacing non-virtualizable instructions with hypercalls communicating with hypervisor for critical operations (time keeping, interrupt handling etc.). Though it lacks benefits provided by full virtualization, it provides low performance overhead which varies with the workload. Eg. Open source Xen project, Vmxnet etc.

2.1.1.1.3 Hardware Assisted Virtualization

The advent of virtualization led the vendors to introduce another root mode below Ring 0 for the VMM to trap sensitive and privileged instructions automatically in hypervisor. The guest state is stored in VM Control Structures (as in Intel VT-x) or VM Control Blocks (as in AMD-V). This rigid model has low software flexibility, high cost of hypervisor to guest transitions and hence has limited applicability.

2.1.1.2 Memory Virtualization

Sharing and dynamic allocation of physical memory to virtual machines constitutes memory virtualization. Similar to memory management provided by modern OSes, there is an additional level of memory owing to virtualization of MMU. The VMM uses TLB hardware to maintain shadow page tables for a direct lookup during guest OS’s VM to

PM mapping. The overhead in this process can be minimized by hardware assist methods.

2.1.1.3 Device and I/O virtualization

Software based I/O virtualization manages routing of I/O requests between virtual devices and shared physical hardware enabling a plethora of features with simple management. Virtual devices can interact with each other seamlessly without straining physical resources and can be migrated or manipulated spontaneously. The hypervisor standardizes all virtual devices enabling portability across platforms due to compatible configuration of virtual hardware on any physical hardware.

2.1.2 Benefits of Virtualization

Virtualization offers many benefits over legacy systems (E. M. J. N. Matthews 2008) which can be enumerated as below:

- **Ease of debugging and Sandboxing:** Developers can test new OS's on stable hosts and debug their code more effectively owing to system isolation. Such "sandboxed" guests can be used to test new worms/viruses, updated features or experimental softwares before being introduced in production systems. Recovery is faster by using saved versions of guests.

- **Load Balancing by relocation:** Guests can be migrated between systems to balance the workload and aggregate resources efficiently

- **Server consolidation:** A single physical machine may run many guest VMs, each having its own root access and authority to choose kinds of applications/services to run without collaborating with other users. This also maximizes the utilization of computing power of a multi-core/multi-processor machine. Booting an OS is as simple as starting an application.

- **Lower cost of ownership:** Optimum utilization of limited physical resources reduces total cost of ownership of companies and also lowers costs to train employees due to minimal reconfiguration of systems. Users can execute legacy products in a secure environment on new architectures saving on costs of application refinement.

- **Decreased power consumption and Cooling Infrastructure:** Efficient utilization of resources translates to optimum use of power and lowering its costs. Smaller size of infrastructure enables effective cooling of the data centers with less ACs again reducing energy costs.

2.2 XEN Virtualization platform

2.2.1 Approach and Overview

To understand the choice of Xen for this project, we need to glance over the drawbacks of full virtualization over paravirtualization, especially with regards to x86 architecture used in the lab environment (P. Barham 2003). Full virtualization poses problems such as supervisor instructions failing without a trap if executed with insufficient privilege and MMU virtualization which can be resolved at the cost of increased complexity & reduced performance. Further, situations where both real and virtual time support is required (for a guest OS to implement time-sensitive tasks; to handle RTT, TCP timeouts), full virtualization proves ineffective.

Paravirtualization promises better performance at the cost of negligible modifications to the guest OS. The following table lists various fundamental tasks handled in Xen interface:

Memory Management	
Segmentation	Fully-privileged segment descriptors cannot be installed or overlapped with top end of linear address space
Paging	Batched and validated updates by hypervisor are fed to guest OS which has direct access to hardware page tables
CPU	
Protection	Xen runs at higher privilege level than guest OS
Exceptions	Exception handlers need registered descriptor tables with Xen
System Calls	A fast handler installed by guest OS may allow direct calls from application but may not indirect them from Xen into OS
Interrupts	Lightweight event system in place of hardware interrupts
Time	‘Real’ and ‘Virtual’ time maintained on each guest OS
Device I/O	
Network, Disk	Easy access V-devices & asynchronous I/O rings

Figure 3: Table describing resource management in Paravirtualization

2.2.2 Xen Architecture

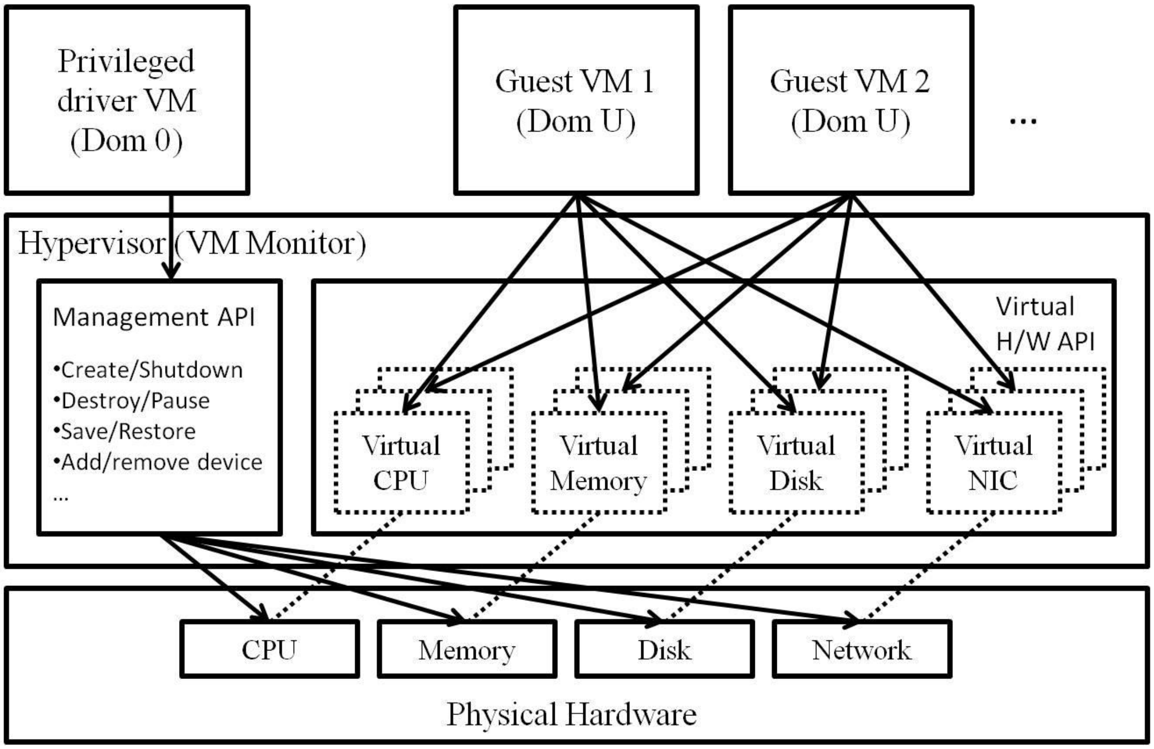


Figure 4: Overview of Xen Architecture

Xen architecture comprises of three main components on the top of physical hardware (E. M. J. N. Matthews 2008) as shown in Figure 3; namely (1) Xen Hypervisor/ VM Monitor, (2) Privileged/Driver Domain and (3) Guest/User Domains

The following sections highlight the architectural features of Xen:

2.2.2.1 Xen hypervisor / Virtual Machine Monitor (VMM):

The layer in direct conjunction with physical hardware providing a virtual interface to the guest domains is the Xen hypervisor (VMM). Here is a summary of the role performed by the VMM:

- Each guest domain receives a specific portion of the physical machine resources. This resource distribution may be arbitrary, equitable or follow some user policy. The VMM may choose to restrict access to some physical device from a guest domain or may even create a non-existent virtual device.

- VMM offers generalized devices to guests. The make of a device is of secondary importance as it is idealized to a general class device; a network device or a block device in case of storage devices. This aids in migration of guests, primarily because the idealized drivers for these devices are managed by Domain 0 (privileged domain).

- VMM is capable of modifying portions of host architecture that are difficult to virtualize. This necessitates changes in guest OS but not user applications.

To carry out these functions, the VMM occupies a privileged position on the system which is provided by means of protection rings discussed in earlier section on paravirtualization. Similar to system calls by apps, guest OSes make hypercalls to hypervisor to request resources and in return receive isolation from other domains and shared resources by asynchronous events like UNIX signals or interrupts.

2.2.2.2 Privileged/Driver Domain (Domain-0):

Domain-0, also called the driver domain, is a special privileged OS that performs administrative tasks for the hypervisor. Launched at boot time, it is the primary tool to manipulate and migrate guest domains between physical machines. Dom0 creates idealized drivers for network/block devices on all guests and communicates with guests through asynchronous shared memory transport.

A backend driver runs on Dom0 providing each generic frontend driver on guests with the illusion that an entire device is dedicated for their use. Any device requests from the frontend are forwarded dynamically to the hardware by the backend, encapsulating them into proper formats. The security and stability of the system is improved by such backend support from Dom0 by resolving sharing & multiple access issues between guests. Due to this reason, Dom0 is usually not overloaded with user applications limiting its scope to efficient DomU management.

Dom0 yields two tools for VM management:

- **XEND (Xen Daemon):** This is a critical process of Xen running as root in Dom0. It provides an interface called xm (Xen Management) to the end-users in order to create, shut down or manipulate guest domains and their resource configuration. Both xend and xm being Python scripts are employed to handle, sequence and validate requests from multiple sources and pass them on as hypercalls to the underlying hypervisor. Various logs are produced and maintained by xend in a rotating log file to provide insight into its status. Its configuration file xend-config.sxp can be tweaked to control the different features offered. Refer (E. M. J. N. Matthews 2008) for more information on xend.

- **XENSTORED (Xen Store Daemon):** Xen provides a way to share configuration information among multiple guests by maintaining a common database called XenStore for the domains to read/write enabling communication. It is mostly used to control devices in guest domains, carry out atomic operations like read/write keys and handle configuration status between backend & frontend drivers. Refer (E. M. J. N. Matthews 2008) for more information on xenstored.

2.2.2.3 Guest/User Domain (DomU)

The modified operating systems running atop a VMM managed by Dom0 are called the guest domains or DomU. Various user applications run on user domains interacting with the frontend device drivers without any changes to the API provided. The xm tool has a set of actions aimed to control the behavior of these guest VMs.

2.2.3 Xen Management & Monitoring tools

2.2.3.1 Xm tool

The xm command is used for administration of Xen guest domains. Management of different devices, manipulation of guest VMs and reconfiguration of different resources like CPU, memory etc. and policy design are some main function of xm (grouped by category in Figure 5). System information can also be retrieved for debugging or monitoring purposes. The given map groups all the commands provided in xm under specific categories.

2.2.3.2 XenTop

Xen provides a xentop program similar to ‘top’ feature provided in Linux. It runs an interactive program showing all running VMs in a table format with resource consumption statistics. It is a user friendly tool to quickly grasp the status of the system and to watch the effect of certain actions on it. The list of information provided is dynamic and adjusts itself with new guests being created or shut down. It can include percentage of CPU and memory usage, network traffic etc. Some hot keys provided underneath the table can be used to set update time, toggle network or virtual block devices display, VCPU information and sorting the data.

2.2.3.3 XenMon

XenMon is QoS Monitoring and Performance profiling tool built by a team of professionals from HP Labs (D. Gupta 2005) for assistance in managing tasks such as support for policy-based resource allocation & VM migration, QoS Provisioning and admission control of VMs. The tool displays shared resource access and scheduling information of all VMs running on the system.

The essential components of xenmon are as follows:

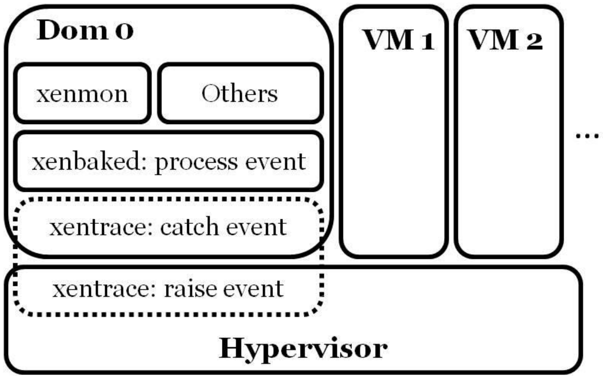


Figure 5: XenMon Internal Processes

- **Xentrace:** This is a small event generation functionality embedded in Xen using which events can be raised at requisite control points in hypervisor along with some associated attributes.

- **Xenbaked:** The events generated by xentrace are further refined in order to be presented to the end user in a simple manner. Different domain events like sleep or wake may be gelled together and reformatted to determine time durations or percentages of resource consumption. A single xenbaked instance can be read and displayed by different xenmon front-end scripts via shared memory region. It is configurable at a large extent accepting user input about sampling frequency, length of history to be stored, samples to be counted etc.

- **Xenmon:** This is a portable front-end python script to display and log data for processing. Though it provides a negligible overhead of 1-2 %, it can be replaced with faster scripts in high level languages.

Xenmon keeps track of different metrics aggregated over a period of domain execution, one second or last 10 seconds. The following table enumerates the same:

CPU Usage	Percentage of CPU run-time used by a domain
Block time	Percentage of time spent by domain blocked on I/O event
Wait time	Percentage of time domain waited for its turn on run queue
Execution count	Frequency of domain scheduling on CPU
I/O count	Number of memory page flips between guest and Dom0 (rough measure of I/O requested by domain)

Figure 6: Table listing XenMon metrics

2.3 Related Work

Resource consolidation and VM management on an array of system configurations has always been a topic of substantial research. The idea of such a study germinated from research undertaken by Ohio State University on High Performance VM Migration using RDMA on InfiniBand (W. Huang 2007). It aimed to utilize low software overhead and one sided nature of RDMA to improve the efficiency of VM migration using Xen on IB architecture. We further delved deeper into the energy and performance efficiency in a data center environment. (A. Bertl 2009) and (H. Abdelsalam 2009) suggest the need for such efficiency in IT, techniques for improvement and current challenges in cloud computing. The management problem is formulated as an optimization problem to minimize overall energy consumption in cloud. (S. Srikantaiah 2008) analyze relationships between energy consumption, resource utilization and consolidated workload performance.

Recent techniques focus more on resource provisioning under dynamic workload conditions. (D. A. Menasce 2006) propose an autonomic controller which dynamically allocates CPUs in virtualized environment with varying workload levels by optimization of a global utility function. We collaborated with researchers from Drexel on a novel idea (P. Grandis 2009) of elicitation and utilization of utility functions for self-assessment of Autonomic Applications encouraging us to map the different environment metrics to good/bad states and take proactive decisions about management of server VMs. (R. Nathuji 2007) consider the heterogeneity of underlying platforms while effectively mapping the workloads to best fit the environment. (J. O. Kephart 2007) address coordination between multiple autonomic managers for power/performance trade-offs using utility functions. (D. Gmach 2008) take an integrated approach to resource pool management by making workload placement controller reactive to changes.

The experiment framework used in this thesis runs parallel to another study conducted at University of Florida on VM migration in support of reservation of cluster resources (M. Zhao 2007). (P. Apparao 2008) present a study on the impact of consolidating several applications on a single server running Xen. (B. Sotomayor 2006) studies overhead issues in VM redistribution. (Y. Song 2007) propose an adaptive dynamic scheme for adjusting resources like CPU or memory between virtual machines on a single server for effective sharing. (S. Kumar 2009) present VManage, a practical coordination approach which loosely couples virtualization and platform management to improve energy savings and reduce VM migrations. Finally, (W. Voorsluys 2009) present a performance evaluation on effects of live migration of VMs on application performance.

3 RESEARCH GOALS

The purpose of the research undertaken is threefold. Firstly, we aim to understand the underlying support available for virtual machine migration and pursue new technologies or abstractions to improve efficiency and speed of the data transfer. In layman terms, we strive to find better ways for how to perform migration. Secondly, we carefully evaluate all the resources used by VMs for proper functioning and study the synchronization and multiplexing processes underneath which delineate when and where to migrate a virtual machine. Our primary concern here is to take decisions about manipulation of a domain before it crosses a certain threshold. Finally, we need to know what is to be done to a running VM (suspend, save, migrate or shutdown) so as to resolve the issue at hand. The following sections illuminate the different approaches taken to achieve our goals.

3.1 RDMA-based approach for VM Migration

At the outset, we began with an extensive study of the Remote Direct Memory Access (RDMA) technology to expedite the underlying process of VM migration. RDMA permits high throughput low latency networking, enabling zero-copy mechanism for fast data transfer using Read/Write operations. Different architectures such as InfiniBand, iWarp, VIA etc. provide RDMA infrastructure whose features can be utilized by an abstraction provided over the hardware. We used the Portals library provided by Sandia National Laboratories, and modified an existing DART implementation (C. Docan 2007) to work in a Xen environment.

DART (Decoupled Asynchronous Remote Transfers) is composed of three components: Client, Server and Remote Receiver. An application signals the DART server with information about a transfer and continues computation. The client runs in-line with the application and provides hits for transfers to the server. The server is responsible for scheduling and transferring application data in the background. It extracts data from compute nodes using RDMA transfers based on client notifications.

A Xen VM migration procedure typically works as follows (P. Barham 2003):

- **Pre-copy stage:** All memory pages used by guests are mapped from source to pre-allocated memory address spaces on sink by helper processes on both side Dom0.
- **Actual transfer:** Memory pages are sent to destination over TCP/IP sockets, taking into account careful translation of machine dependent addresses in page tables.
- **Termination stage:** After the transfer is complete, guest OS on source machine is discarded and execution is resumed on destination.

For seamless working, it is advisable to have VM disk image on NFS and VM configuration file transferred as well.

We attempted to buttress the second transfer stage of migration described above, with high throughput DART mechanism. For comparison, we analyzed the data transfer of a typical virtual machine size file (say 512 MB) by running DART over TCP/IP. The results are portrayed in section 5.1. Further, we ported the C library of DART into python scripts embedded in Xen for migration to perform the same operation on a real guest VM checkpoint file.

3.2 Utility-based approach to VM Management

We further proceeded to our second goal of examining experimental data exhaustively to draw conclusions about when to manipulate a running guest VM. This is a non-analytic approach to self-assessment of a system for autonomic computing, highlighted in (P. Grandis 2009). The idea is to leverage utility functions at the level of applications running on a virtual machine by collecting exhaustive samples of runtime data about relevant application (VM in our case) attributes to design a utility function and embed code within the Xen library to take online decisions about an action to be performed. Here is a block diagram conveying the same idea:

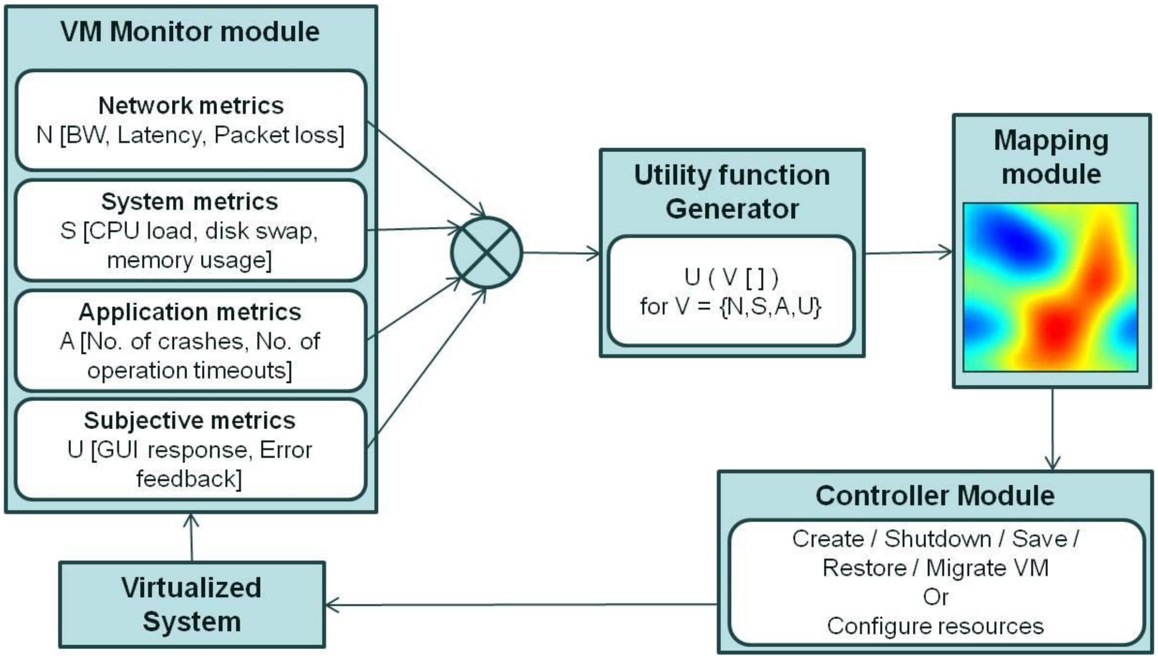

VM Monitor module
Network metrics
N [BW, Latency, Packet loss]
System metrics
S [CPU load, disk swap, memory usage]
Application metrics
A [No. of crashes, No. of operation timeouts]
Subjective metrics
U [GUI response, Error feedback]
Utility function Generator
U (V [ ])
for V = {N,S,A,U}
Mapping module
Controller Module
Create / Shutdown / Save / Restore / Migrate VM
Or
Configure resources
Virtualized System

Figure 7: Block Diagram for Utility based VM Management

The approach can be described as follows:

- **Synthesize application utility on a single VM with respect to various benchmarks running on a host machine**

Consider a web server-client test scenario. Monitor response time of the server which runs on a single VM allotted with fixed share of resources. Let the application use 100% of its allotted resources. We find the correlation between response time and network, CPU, memory parameters. One can get a 2-D or a 3-D graph of response time versus percentage of CPU allotted and/or memory allotted to VM. A region/surface of “good” performance can be noted down, and threshold values set for independent parameters, depending on quality of service desired by the application or its SLA.

- **Design model of interference framing utility functions for multiple VMs**

Add another VM to above scenario with another disjoint set of resources (for instance, let VM1 use VCPU(1,2) and VM2 use VCPU(3,4) on the quad-core machine), and study the interference on first VM. We can do the same with joint set of resources (Both VMs using all 4 logical CPUs of a quad-core machine). Here, we study the effect on VM1 workload checking if the VM lies in its “good” set or surface or how close it moves to the set boundary. Resources like CPU or memory can be re-configured dynamically so as to bring the other VM back within closed region. We can vary number of VMs and distribution of resources to keep all VMs within certain region of optimal performance by studying old VMs position in the map.

- **Elicit an action to be taken based on the model's outputs.**

Resources can be re-allocated, VMs can be shutdown, created, save or restored; or VMs could be migrated to another free machine if re-configurations are not possible (in case of critical VMs). We can build a cross layer management module (i.e. to record metrics at VM level and study their impact on utility and action taken).

3.3 Analysis of the performance efficiency using resource-intensive benchmarks

To achieve the first objective of our three-pronged goal described above, we carried out exhaustive stress tests on guest VMs for different application scenarios. These tests sought to quantify the performance isolation properties of virtualization systems by studying an overloaded guest VM's impact on its neighboring virtual machines and system resources. By gathering extensive results of a VM's behavior under extreme conditions, we may design a model of interference to suggest reconciliatory measures.

3.3.1 ApacheBench with XenTop monitoring tool

We first studied the usage of network resources by running Apache web servers in different VMs on a Xen platform. The servers received a constant workload from web clients on a remote machine running ApacheBench. For each workload, we explored different CPU configurations by adjusting the following parameters:

- Number of VCPUs shared between VMs
- Number of VCPUs pinned to physical CPUs per VM
- Absolute and Relative weight of CPUs allotted per VM

The web server response time was monitored and normalized w.r.t. its value at idle time. The basic monitoring tool ‘Xentop’ was run on Dom-0 of server machine.

3.3.2 Isolation Benchmark Suite

The isolation suite quantifies the level to which a virtualized system affects the impact of an abnormally operating virtual machine on other stable machines (W. H. J. N. Matthews 2007). It takes into account all the resources, forming six different stress tests (CPU, memory, disk, network intensive and a fork bomb). The usual procedure is to run a set of virtual machines engaged in different tasks and run one stress test on any single guest. The repercussions of the misbehaving VM are noted and the service quality degradation is studied as a percentage normalized to regular behavior. In our case, we ran Apache web servers on all guest VMs maintaining a production workload and then proceeded to run stress tests.

The earlier two tests dealt with constant workloads, while the latter two will highlight virtualized environments with varying workloads, resource re-configuration on a finer granularity and a new set of monitoring tools.

3.3.3 Httperf benchmark with XenMon monitoring tool

We repeated the web server tests with a new benchmark and monitoring tool, but focused our attention on CPU scheduling in Xen. Httperf was an improvement over ApacheBench in many aspects, especially its ability to provide varying workload to the web servers installed on VMs. While 3.3.1 concentrated on running multiple guest domains on server side and using ‘ab’ to send a sizeable workload, this time we took into account the role of Dom-0 in the model of interference and executed web server on one guest domain.

Further, we narrowed down the number of CPUs used by both domains to one, using 3 different set of configurations shown in 4.4.1. The credit scheduler in Xen was used for this purpose.

As a monitoring tool, XenMon proved more effective over XenTop for its deeper insight into CPU scheduling, operation time and I/O behavior. XenTop showcased change in CPU usage percentage, but XenMon took it to a fine granular level by showing the exact percentage of running, waiting and block times of VMs on CPU as well as number of executions per second. Also, XenMon added about 1-2 % overhead on the running system which was an acceptable limit. On the server side, we studied the average connection time and response time of servers for increasing workload and constant CPU configuration. The percentage of CPU utilization was compared against increasing workload by studying blocked, waiting and run time percentages for both domains.

3.3.4 CPU Frequency Scaling with DVFS

To delve deeper into the CPU performance, we turned to frequency scaling using the acpi-cpufreq module. This driver was chosen over the p4-clockmod for not only reducing the clock frequency of a CPU, but also reducing its voltage. This results in lower power consumption and heat output for each unit reduction in performance.

For our system, we had the following features at our disposal in acpi-cpufreq module:

Available CPU cores: 4

Available frequency steps: 2.40 GHz, 2.13 GHz, 1.87 GHz, 1.60 GHz

Available governors: ondemand, userspace, performance

The governor ‘ondemand’ could decide which speed to use with the 1.6-2.4 GHz range.

Working further on our available configuration of 2 domains (controller and guest domain), we pinned both domains to one or more CPUs and tried different CPU frequency combinations studying their effect on response time of web servers on guest virtual machines.

3.3.5 Compute-intensive FFTW benchmark

In our search for computationally intensive benchmarks to subject our system to, FFTW proved to be a formidable choice owing to the flexibility it offered in controlling the size of data sets under computation. The idea of using this benchmark was derived from an earlier work (I. Kulkarni 2009) at CAC where the same was effectively used for dealing with HPC workloads. The experiments designed are analogous to using forkbomb and CPU-intensive scripts in the Isolation benchmark suite in 3.3.2, but offer more user control by varying matrix sizes.

3.4 Analysis of Parallelization and Location Dependency in VM management

Until now, our prime focus was on VM migration and the way resources are affected in the process. VMs provided a primitive for transparently redistributing workloads through migration; however the entire process of creating, powering down, saving, migrating VMs on the run can be quite time consuming and needs careful estimation. Henceforth, we tried to build a more comprehensive model that could characterize all essential VM operations and chart its performance to enable decision making. For instance, such an analysis would be beneficial in a scenario where one or more physical servers need to be emptied to launch a cluster of VMs for dedicated execution of parallel tasks.

We considered the problem of resource reservation for single guest OS instances, as well as VM clusters similar to the approach in (M. Zhao 2007), buttressed with more metrics studied and all-round analysis. We embedded special UNIX scripts to automate the experiments and used both our monitoring tools to record system behavior. In the following sections, we start with different set of operations (E. M. J. N. Matthews 2008) on an increasing number of VMs with images on NFS or local disk.

3.4.1 Create and Shutdown

Create and Shutdown are the basic operations every VM faces in its lifetime depending on its requirement for managing varying workloads. There are different factors which may affect the speed of the operations including location of the disk image (NFS or local file system), number of CPUs allotted to every VM, maximum number of VMs to be operated upon at a time etc. We take into account a few prominent parameters and create our set of experiments as explained in Chapter 4.

3.4.2 Save and Restore

At the heart of any migration is the ability to fully represent the state of a guest. When a guest virtual machine is completely shut down, this is trivial. An inactive Xen guest is completely defined by its file system image(s), configuration file, and its operating system kernel. Clearly, a guest could be cloned or even moved to another physical machine by making copies of these files. Backup of a guest can be accomplished in this way. A guest that is active in execution, on the other hand, is a more complicated matter. While a guest is running, saving its state additionally involves creating a snapshot of its memory, device I/O states, open network connections, and the contents of its virtual CPU registers. Xen can save this state information to disk or transfer it over the network, which allows for both backup and migration of VMs.

This idea of saving the state of a running guest is similar to the hibernation feature on many personal computers, especially laptops. In hibernation, a system's state is checkpointed and saved to disk so that the system can park the hard drive heads, power down, and resume its previous state next time it is powered up. In the case of a virtual machine monitor like Xen, a similar facility can be used to checkpoint states to facilitate rollback in the event a guest fails, or to save the state of a guest past the shutdown of the physical machine on which it is running.

Xen provides a domain save and restore facility to handle the suspension of guest VMs to checkpoint files, operated by the ‘xm save’ and ‘xm restore’ commands. When a guest is saved to disk it is suspended, and its resources are deallocated. As in the case of hibernate, ongoing network connections are not preserved. We follow a procedure similar to create/shutdown operations to evaluate the impact of these operations on system resources.

3.4.3 Migrate

With the xm migrate command, Xen supports warm static migration (regular migration), where a running guest is temporarily suspended and then relocated to another physical host, as well as live migration, where a guest may be relocated from one host to another seamlessly, without dropping ongoing network connections and with little client perceptible delay. Live migration is particularly useful when bringing down a physical machine for maintenance. In this case, guests can be relocated to a new physical machine in preparation for the maintenance without a disruption in service. The ability to relocate a guest is also useful for load balancing guests and their resource consumption.

Migration requires a guest's memory contents to be transferred, I/O transactions temporarily quiesced, CPU register states transferred, and network connections rerouted and resumed on the destination host. If a Xen user wants to use Xen's integrated migration support, it is currently necessary to configure guests to access their file systems over network shares that are equally available to both the source and destination host. In regular migration, a guest domain is suspended to easily handle the transfer of guest memory pages that would be continuously changing if the guest was otherwise allowed to continue to run. Suspending the guest also satisfies the need to quiesce I/O.

We do not consider live migration in our set of experiments to reduce complexity of the operations and analyze results in an easier manner. Similar to prior operations, we follow a carefully monitored procedure for migration between two distinct physical machines and study consumption of system resources on both of them.

4 EVALUATION METHODOLOGY

Once we are through with the system installation and verified the availability of sufficient resources and configuration on Xen platform, we can design a set of experiments to perform stress tests, run different workloads and study resource usage at fine granularity. The following sections provide an overview of various benchmarks mentioned in the previous chapter and the general methodology followed to implement aforementioned approaches.

4.1 Overview of Performance Analysis Benchmarks

We used variety of benchmarks to execute CPU, memory, disk and network intensive operations on the physical machines and studied different metrics which provide valuable insight in the behavior of virtualized systems. Here is a short summary of different benchmarks used for the experiments:

4.1.1 ApacheBench (Apache HTTP Server Benchmarking Tool)

ApacheBench or ‘ab’ is a tool for benchmarking Apache HTTP servers (installed on guest VMs in our set-up). It is designed to delineate the performance of a typical Apache Web Server installation on a machine focusing on the number and frequency of requests per second served. The different features of the tool can be explored in Linux by referring to the manual pages for ‘ab’ (man ab). Once the web servers have been installed and running on respective virtual machines, this benchmark can be run from a remote machine by a general procedure highlighted in 3.3.1

4.1.2 Httperf

httperf is a tool to measure web server performance. It speaks the HTTP protocol both in its HTTP/1.0 and HTTP/1.1 flavors and offers a variety of workload generators. While running, it keeps track of a number of performance metrics that are summarized in the form of statistics that are printed at the end of a test run. The most basic operation of httperf is to generate a fixed number of HTTP GET requests and to measure how many replies (responses) came back from the server and at what rate the responses arrived. To obtain correct results, it is necessary to run at most one httperf process per client machine. Also, there should be as few background processes as possible both on the client and server machines. The httperf manual from the website (HP Labs 2008) gives a detailed description of all the options embedded in the operation.

4.1.3 Isolation Benchmark Suite

This benchmark suite is designed to measure the extent to which a virtualized system protects well-behaving VMs from the impact of one or more misbehaving guests on system resources. It includes 6 different tests focusing on CPU, memory, disk, network intensive operations and exponentially rising forking processes. The researchers behind this benchmark have published their results in (W. H. J. N. Matthews 2007).

The given table explains the function of each stress test:

Test	**Description**
CPU intensive	Integer arithmetic performed continuously in tight loop
Fork Bomb	Creates child processes exponentially within a loop
Memory intensive	Loops allocating & touching memory without freeing it
Disk intensive	10 threads of 'Iozone' run with alternate read & write pattern
N/w-intensive(Tx)	60k size packets sent over UDP by 4 threads to outer m/c
N/w-intensive(Rx)	60k size packets received by 4 threads from outer m/c

Figure 8: Table listing function of each Stress Test

4.1.4 FFTW

FFTW is a portable C subroutine library for computing Discrete Fourier Transform in one or more dimensions on input of arbitrary size with real or complex data. Its primary purpose here is to perform compute intensive tasks of increasing complexity on guest VMs so as to measure their behavior under different load conditions. The benchmark provided alongside 'benchFFT' integrates a large number of publicly available FFT implementations and measures their performance and accuracy over a range of transform sizes. The package has been developed over 40 years and run on variety of computer architectures. The tutorial, installation guide and execution procedure is given in (M. Frigo 2005).

4.2 CPU Frequency Scaling

CPUFreq is a tool that can be used to control power consumption in the system by scaling CPU frequency. It allows the clock speed of the processor to be adjusted on the fly so that reduced clock speed translates into low power consumption. The functionality to control the clock speed and time it according to certain events is ingrained in the CPUfreq governor.

Various power levels of the system processor are defined by the governor displaying unique behavior, purpose and suitability according to workload. A central requirement of power management is to grasp the effective optimization of energy consumption pertaining to each system component. The governor studies different tasks performed by the system and configures each component to ensure that its performance mirrors the job at hand. Since lowering power consumption of the system or its component will affect its performance as well, careful evaluation of each configuration needs to be done before putting the system to test.

4.3 Performance Evaluation

4.3.1 Experimental Framework

The central framework for all the described experiments is as follows:

- Two Dell PowerEdge R200 server machines (PM-1 and PM-2) with Intel Xeon quad-core 2.4 GHz CPU and 4 GB RAM

- CentOS 5.2 as Domain-0, using Linux 2.6.18-92 kernel with Xen 3.0 version

- 10 paravirtualized guest machines (VM-1 to VM-10) on each physical server (Specifications: CentOS 5.2 distribution, 512 MB RAM, 4 VCPUs, 8 GB disk space)

Using ‘xm’ functionality along with the previous benchmarks and tools, we carried out the subsequent experiments to analyze VM behavior in the above setting.

4.3.2 Experimental Procedures

4.3.2.1 Using ApacheBench

This procedure tests the client-server interaction and performance (server response time) with a constant workload fed to a physical server running to 2 virtual machines with Apache2.0. We adjust CPU settings for the quad-core server per VM and plot resultant response time.

- Run ‘xentop’ on PM-2 (server machine) with 1 sec delay between batch updates and log the output in a file for up to 3 minutes (180 sec).
    - `xentop –b –d 1 –i 180 > logfile.txt`
- Create 2 VMs on PM-2, each running an instance of Apache web server
- Adjust CPU settings for both VMs in following manner:
    - Set number of VCPUs per VM (Xm vcpu-set <Dom> <No. of VCPUs>)
    - Pin VCPUs to set of PCPUs (Xm vcpu-pin <domain> <VCPU> <CPUs>)
    - Set CPU credit sched. (Xm sched-credit –d <domain> -w <weight> -c <cap>)
- Run ‘ab’ benchmark on client PM-1 for different values of total number of requests sent (10k-100k) and concurrency of the requests (5, 50, 500) enabling ‘KeepAlive feature. The requests are sent to the web server on each VM.
    - `ab -k  -n <requests> -c <concurrency>  http://serv-addr/page.html > results.txt`

- Record the average response time for every CPU setting and plot it.
- After each test, reboot the web server and repeat procedure.

The results are discussed in next chapter.

4.3.2.2 Using Isolation Benchmark suite

These set of tests account for all resources of physical server by stressing one component (CPU, memory, disk or network bandwidth) at a time on one running guest VM, and studying its effect on the behavior of other guest VMs. The VMs run web servers maintaining constant workloads.

- Start a set of ‘n’ stable virtual machines on the same physical machine.
- Start Apache web servers on each machine receiving a regular production workload.
- Record baseline response time for the initial configuration.
- Run xentop on Dom-0 of the physical machine.
- Execute one of the stress test programs on one VM.
- Record the response time of the resultant configuration.

4.3.2.3 Using Httperf

The following procedure is similar to 4.3.2.1 except that it deals with varying workloads and records percentage of CPU utilization along with web server performance on one guest. The use of XenMon tool gives deeper insights in CPU allotment policies of credit scheduler.

- Create guest VM-1 on server machine PM-2 and run Apache web server.
- Set CPU configuration for VM-1 and Dom-0 as follows:

  - Set no. of VCPUs as 1 for each domain (uniprocessor)
  - Pin VCPU 0 to physical CPU 0
  - Set weights <10/100>, <100/100>, <100/10> for 3 configurations of <Dom-0/Dom-1> called Conf_0.1, Conf_1, Conf_10 respectively.
- Run httperf client on PM-1, sending increasing workload to server as shown:
  - 50 req/sec to 500 req/sec in steps of 50
  - Monitor response time & average connection time for 60 sec duration
- Run xenmon.py process on glitch to monitor CPU behavior for 30 sec.
- Record Blocked time, Waiting time, CPU Utilization & Exec count/sec
- Repeat the steps for each CPU configuration and plot results.

4.3.2.4 Using Frequency Scaling

The given method adds another control parameter to our existing set in the form of CPU frequency. We study effects of scaling of individual CPU core frequencies on controller and guest domain's computing performance.

- Pin Dom-0 and VM-1 to one or more CPUs.
- Select desired governor using (Cpufreq-selector –governor <type>)
- Set desired frequency using (Cpufreq-set –c –f <frequency>)
- Verify the changed configuration by (Cpufreq-info)
- Repeat prior web server experimental procedure using httperf and xenmon.
- Record results and repeat for different frequency settings

4.3.2.5 Using FFTW

This methodology delves into ALU performance by carrying out complex FFT operations on matrices with user-defined sizes, and notes the changes for increasing VM number.

- Install FFTW benchmark on guest VMs.
- Record the baseline CPU utilization using XenMon
- Run ‘fftw’ on ‘n’ guests for varying matrix sizes.
- Record the new CPU utilization keeping the configuration same.
- Repeat the procedure for increasing number of guest VMs.
- Optionally, change CPU proximity for different VMs and record results.

4.3.2.6 Create, Shutdown, Save, Restore, Migrate operations

This last procedure studies transition from sequential to parallel VM operations on a uniprocessor configuration with VM images switching between NFS and non-NFS.

- Install disk images on the local file system of the PM-1.
- Modify the config script for every VM to be manipulated to point to the image.
- Set Dom-0 to use ’v’ VCPUs (v = 1 to 4)
- Run monitoring scripts
- Create & shutdown ‘n’ VMs (n = 1 to 5)
    - Set VCPU=1 on after every VM creation
    - Repeat 5 times to find average metric value
- Stop scripts, study effect on Dom-0
- Record required 5 metrics (% run, block, wait times, executions/sec, operation time)

- Repeat the above set of steps after copying images on NFS and modifying config files accordingly.

For migrate, run the monitoring scripts on PM-2 as well and record the CPU behavior while receiving the migrated VM.

The above set of experiments gave us an interesting set of results which we will peruse in the next chapter. Some non-conclusive results have been omitted from the document.

5 RESULTS

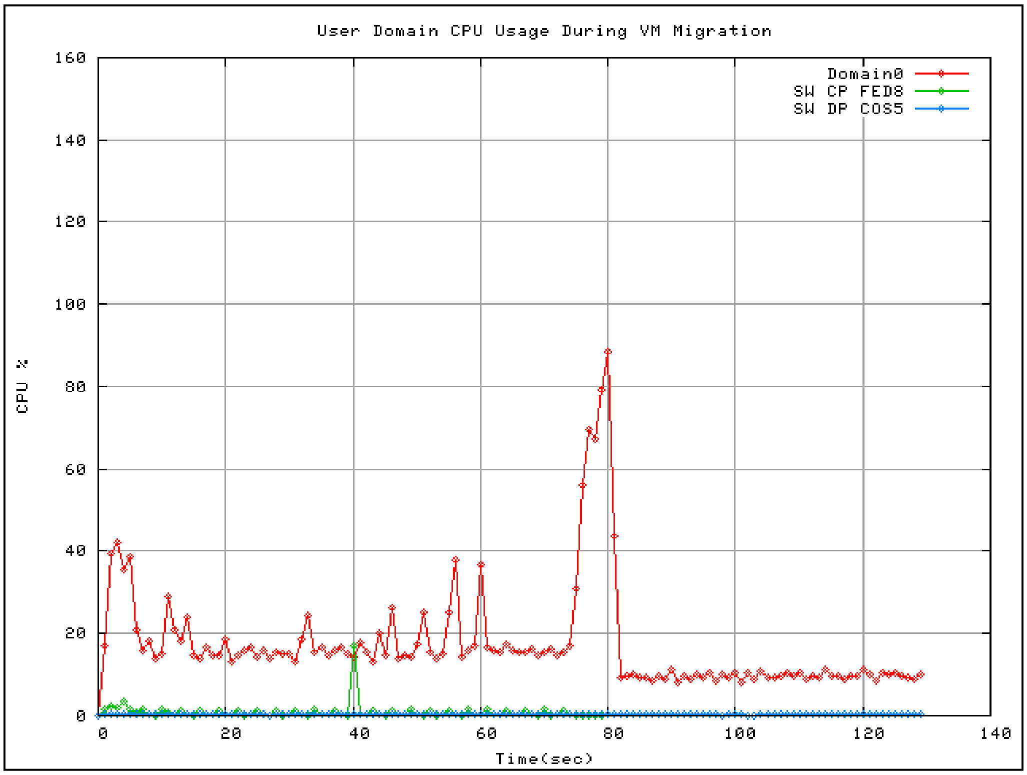
This chapter presents the observations from the different research approaches applied in Chapter 3 and the results obtained owing to the methodologies applied in Chapter 4. Each sub-section will present the results corresponding to sections 3.1, 3.2 and 4.3 in graphical or tabular format, explaining the correlation in values and resultant conclusions.

5.1 Use of RDMA for VM migration

As explained in section 3.1, we present the results obtained by migration of two guest virtual machines checkpoint files; namely Domain-Fed-8 (running Fedora 8 distro) and Domain-Cos-5 (running CentOS 5 distro), over an Ethernet connection using DART between two physical machines running CentOS 5. Domain-0 is the controller domain on the source host.

Xen platform inherently uses python scripts to perform most of its functionality such as creating, saving, migrating guest VMs etc. Hence, it is natural to embed our RDMA abstraction within Python to offer ‘Migrate using RDMA’ facility to end user. The ‘migrate’ script in Xen uses socket interfaces to transfer the VM checkpoint file and config file between two machines and presence of VM image on NFS does not require relocation of the disk image. We integrated our DART implementation over TCP/IP with Python using a ‘ctypes’ library and carried out VM file transfer by running corresponding modified Python script.

The graphs in Figure 10 show the percentage of CPU utilized by the three domains (Dom-0 and 2 guest VMs) during native Xen migration and DART migration. We can see that both guest VMs contribution to CPU activity is negligible. Dom-0, on the other hand, registers high usage while migrating both VMs, for approximately 8 sec. This shows that Dom-0, being the driver domain, is entirely responsible for VM migration and may not be able to handle conflicting programs during VM transfer. Secondly, using RDMA abstraction through DART over TCP/IP does not improve speed of migration since it does not function close to hardware and has a large performance overhead operating over TCP. The larger CPU utilization spike in DART migration signifies the overhead issue.



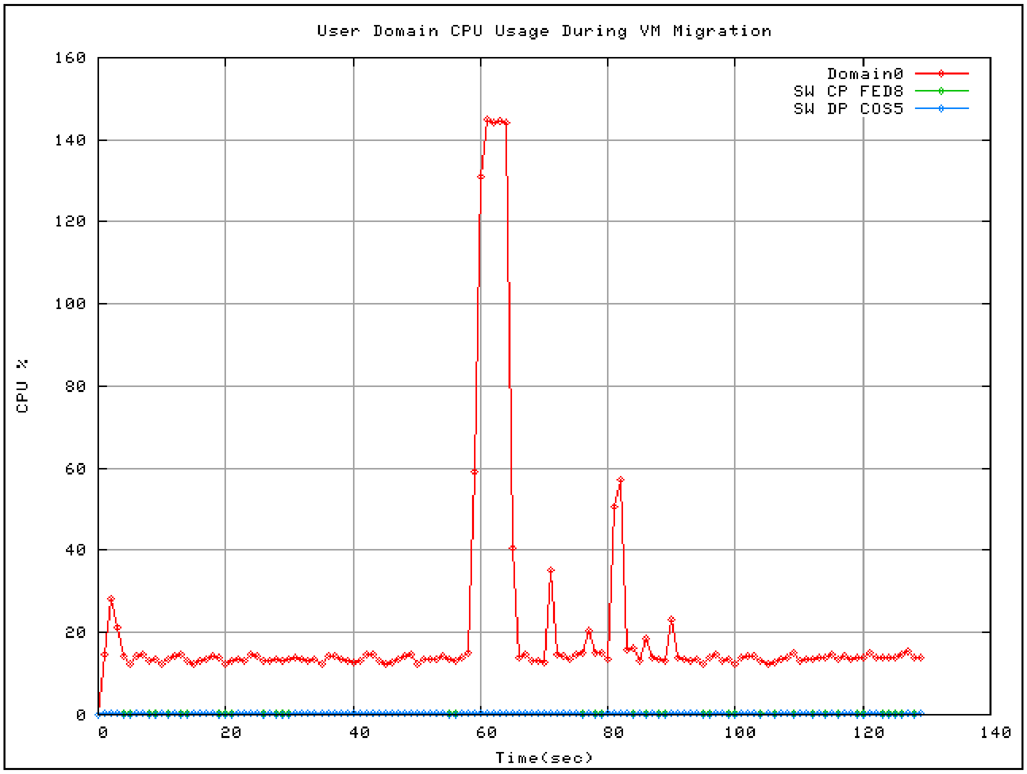


Figure 9: Domain CPU Utlization during (a) Native Migration, (b) DART Migration

Since, DART layer over the network layers is not beneficial, we attempted to run DART directly over InfiniBand cable to make use of hardware infrastructure. Unfortunately, the Host Channel Adaptors (HCAs) on both servers were unequipped to run Xen kernel and modifications to the kernel are essential for VM migration. Such support is sought for in near future.

5.2 Web Server Performance using ApacheBench against CPU scheduling

The following results portray the effect of fine tuning of CPU resources allotted to guest VMs on the performance of web servers running on them.

Figure 11(a) shows a 3D graph of the effect of number of VCPUs allotted to 2 guest VMs on the response time (normalized w.r.t. baseline response) of web servers. The green and red areas depict that the response time reduced than its standard value when VM1 had two or less CPUs at its disposal. Further, the servers registered better throughput when either of the two VMs had half of the original number of CPUs (2 in our case of quad-core machine).

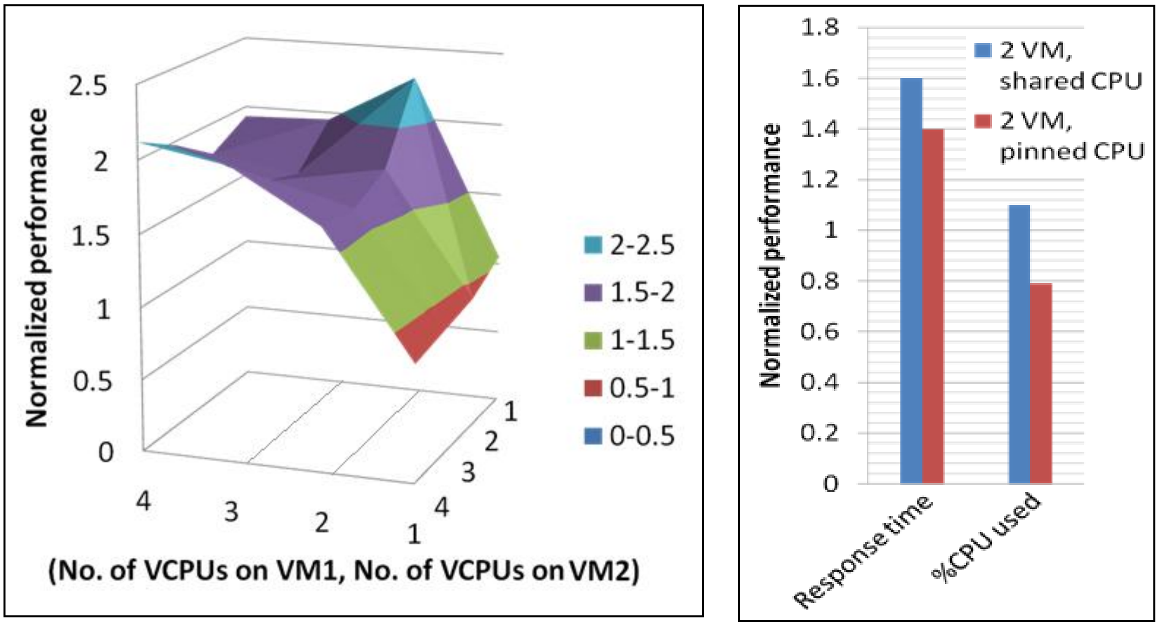
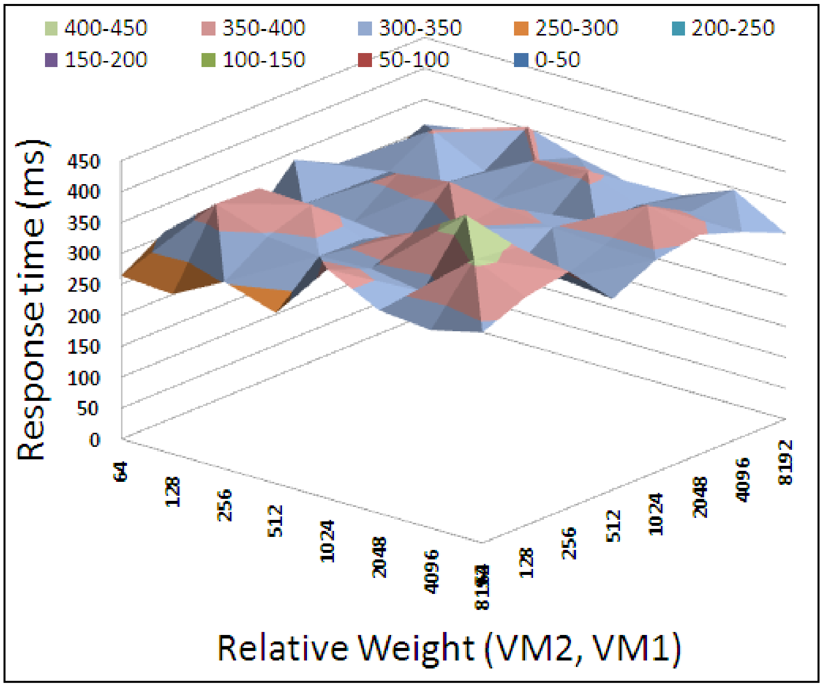


Figure 10: (a) Effect of VCPU no. per VM, (b) Effect of CPU pinning; on response time

Figure 11 (b) shows the effect of pinning of VCPUs to logical CPUs on response time and percentage of CPU utilization. The blue bar signifies two VCPUs per VM sharing all the logical cores of CPU; while the red bar denotes two VCPUs pinned down to two physical cores per VM. Though response time increases in both cases, owing to the increased network traffic; CPU utilization reduces in the pinned case. As long as the number of VMs running is less than number of CPU cores available, pinning CPUs helps reduce response time against sharing all cores.

The next set of figures show a more granular control over CPU utilization per VM by tweaking the weight and cap values of credit scheduler. Figure 12 (a) is a 3D graph depicting the variation of response time of web servers on 2 guest VMs, as the relative weight allotted to each is varied from 64 to 8192 credits. For eg, if VM-1 has 64 credits and VM-2 has 256 credits, VM-1 will receive one-fifth (64/(64+256)) of the total CPU share available. Figure 12 (b) portrays the variation of response time against the absolute share of CPU (cap percentage) allotted to both VMs. For eg, a cap of 200% allotted to VM-1 allows it to use 2 VCPUs out of the 4 possible.

The variation in relative CPU weight allotted to both VMs does not affect the performance significantly. In second case, allotting less than an entire CPU to VM-1 doubles the response time. It is also observed that allotting 100% or 300% of CPU share to VM-1 reduces response time to the same level as a single guest VM would incur.



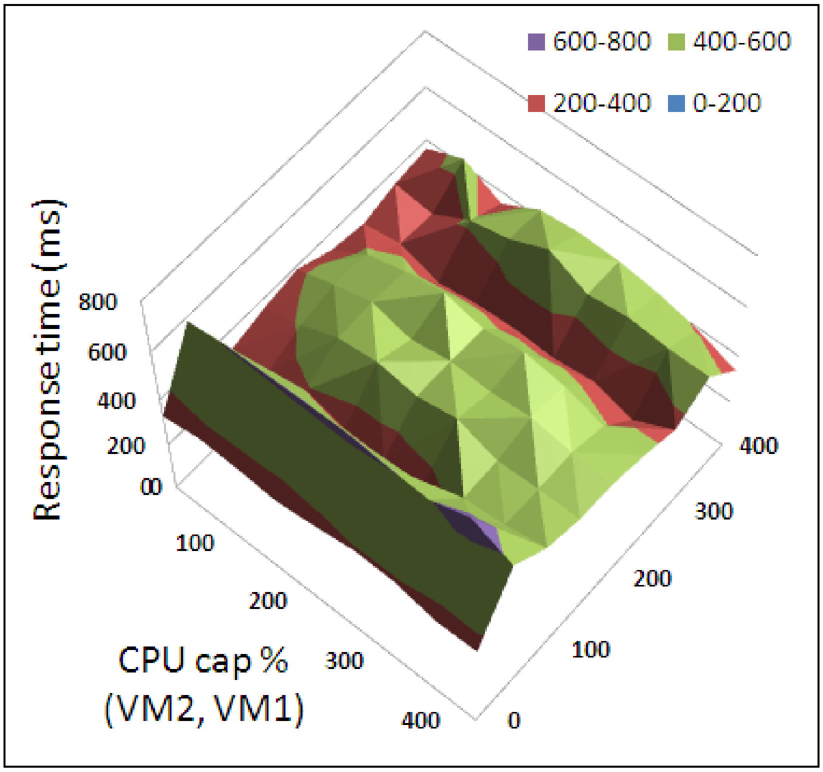


**Figure 11: (a) Effect of relative CPU weight,
(b) Effect of absolute CPU weight on response**

5.3 Web server performance and CPU Utilization using Httperf

The following results are derived from studying the VM interference between guest and controller domain similar to the study undertaken by HP [4]. While the team analyzed the older BVT scheduler used in Xen, we used the credit scheduler. These graphs show the web server performance and CPU utilization for three different uniprocessor configurations:

Conf_0.1: Guest domain's CPU share is one-tenth of controller domain's share.

Conf_1: Both domains have equal shares.

Conf_10: Guest domain's CPU share is 10 times that of controller domain's share.

Figure 13(a) shows the average time required to establish a TCP connection with web server for each configuration against an increasing workload. Figure 13(b) displays the average server response time in the same setting. Both the graphs portray that increasing the workload beyond 200 req/sec for Conf_10 and beyond 300 req/sec for Conf_0.1 increase the connection and response times exponentially. These highlight the threshold workloads for a given CPU configuration.

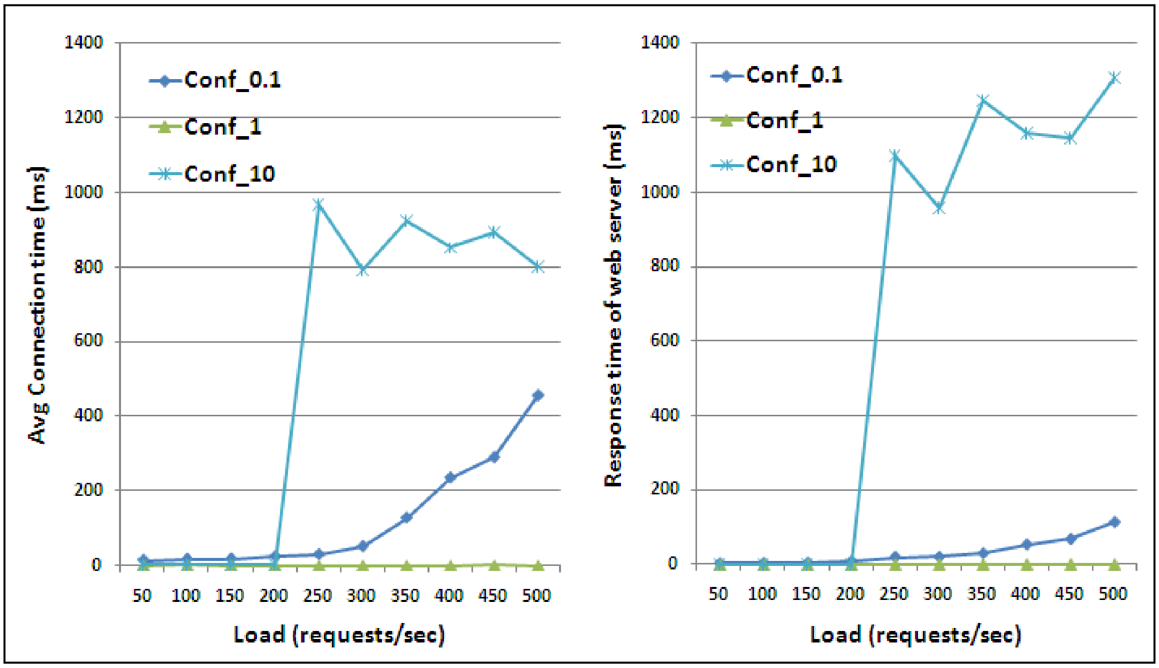


Figure 12: (a) Average connection time, (b) Average response time w.r.t varying workload

Figure 14 shows the comparison between percentage run and blocked times for CPU for both domains in the 3 configurations. In Conf_0.1, run time (%) is higher for guest than controller and increases linearly whereas blocked time decreases similarly. For Conf_1, similar behavior is observed though blocked times are almost equal for both domains. Conf_10 presents an interesting case where % run time increases up to threshold workload and then decreases gradually for both domains. Also, average blocked time %

for guest reduces than that for controller starting from the same workload threshold. The graphs for % wait times and executions per second on the CPU portray results similar to % blocked time, and hence been omitted. The graphs show us that allotting a lower CPU share to controller domain than guest domain(s) adversely affects the web server performance for a varying workload.

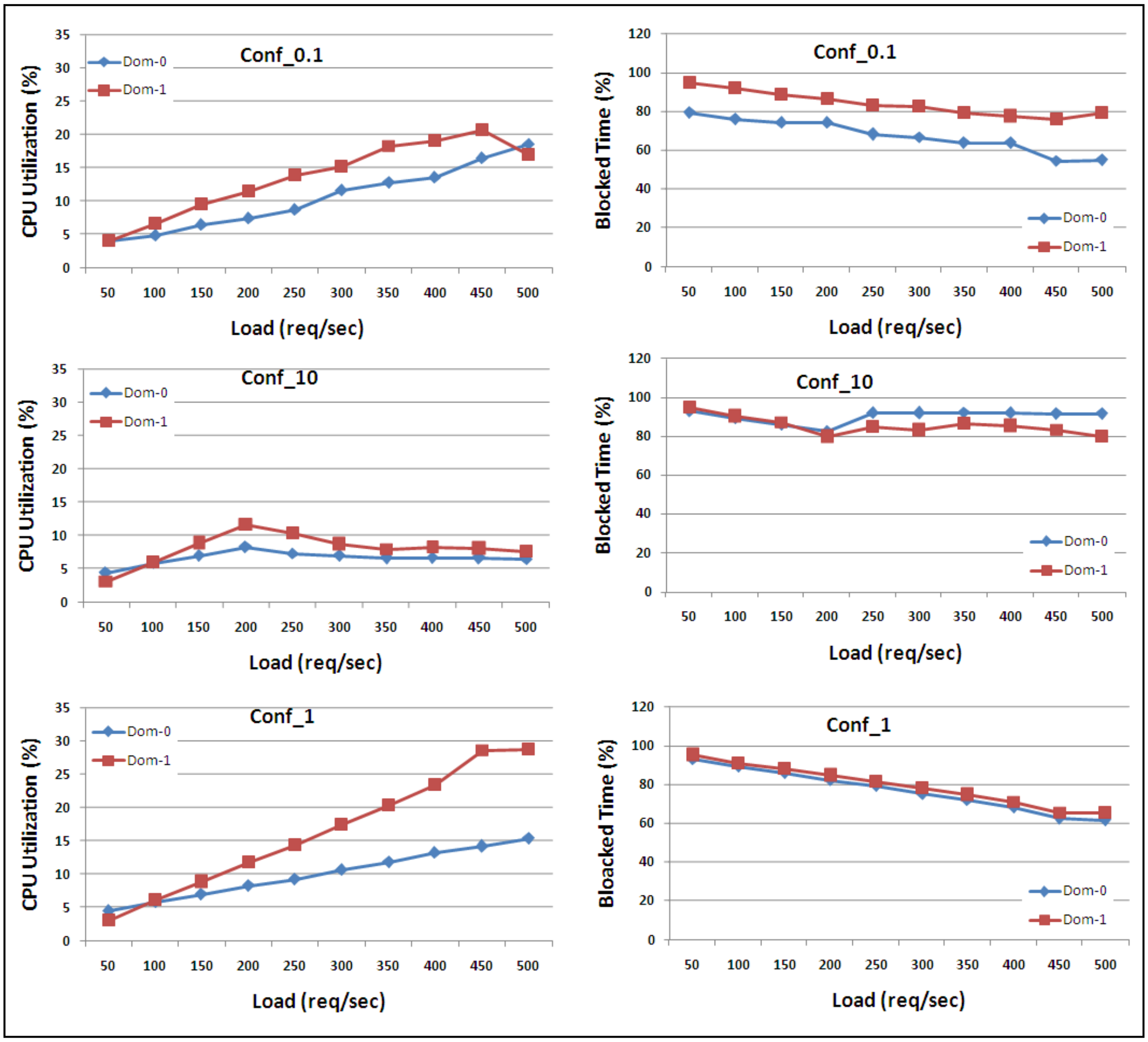


Figure 13: Comparison between % Run time & % Blocked time of 3 CPU configurations

5.4 Effect of Function Parallelization & Image Location on Resource Utilization

Finally, we present the outcome of parallel operations (Create, Shutdown, Save and Restore) on guest VMs whose images are located either on NFS or local disk. The location of images on NFS ensured a complete transfer of guest VMs on destination server since every VM had its configuration file and disk image on NFS and memory pages transferred over as a checkpoint file. On the other hand, using local disk images, though checkpoint files get transferred, applications could not be run on sink side.

Now, we will concentrate on the effect of increasing parallelization of the 4 basic operations on CPU utilization by Dom-0. The sequence of graphs shows the effect of increasing number of VMs on the % run time, % blocked time, % wait time and executions/sec on CPU utilized by Dom-0 (uniprocessor configuration for host machine).

In Figure 15, we note that the Create function requires more % run time than the Shutdown function on Dom-0 which increases gradually with increasing number of VMs. Similar process occurs for % block time, though % wait time increases by an order of magnitude when number of VMs goes from 3 to 4. Save and Restore operations require almost constant % run, block and wait times for increasing VM number though maintaining a constant difference in their values. The normalized values in right hand side graphs show the magnitude of percentage changes in CPU usage. It is interesting to note the constant difference of CPU usage in Save and Restore operations even though both of them deal with archiving and restoring of memory pages from checkpoint file. The high dependence of Create operation’s CPU usage on VM number against Shut down for the same VM number is also equally noteworthy.

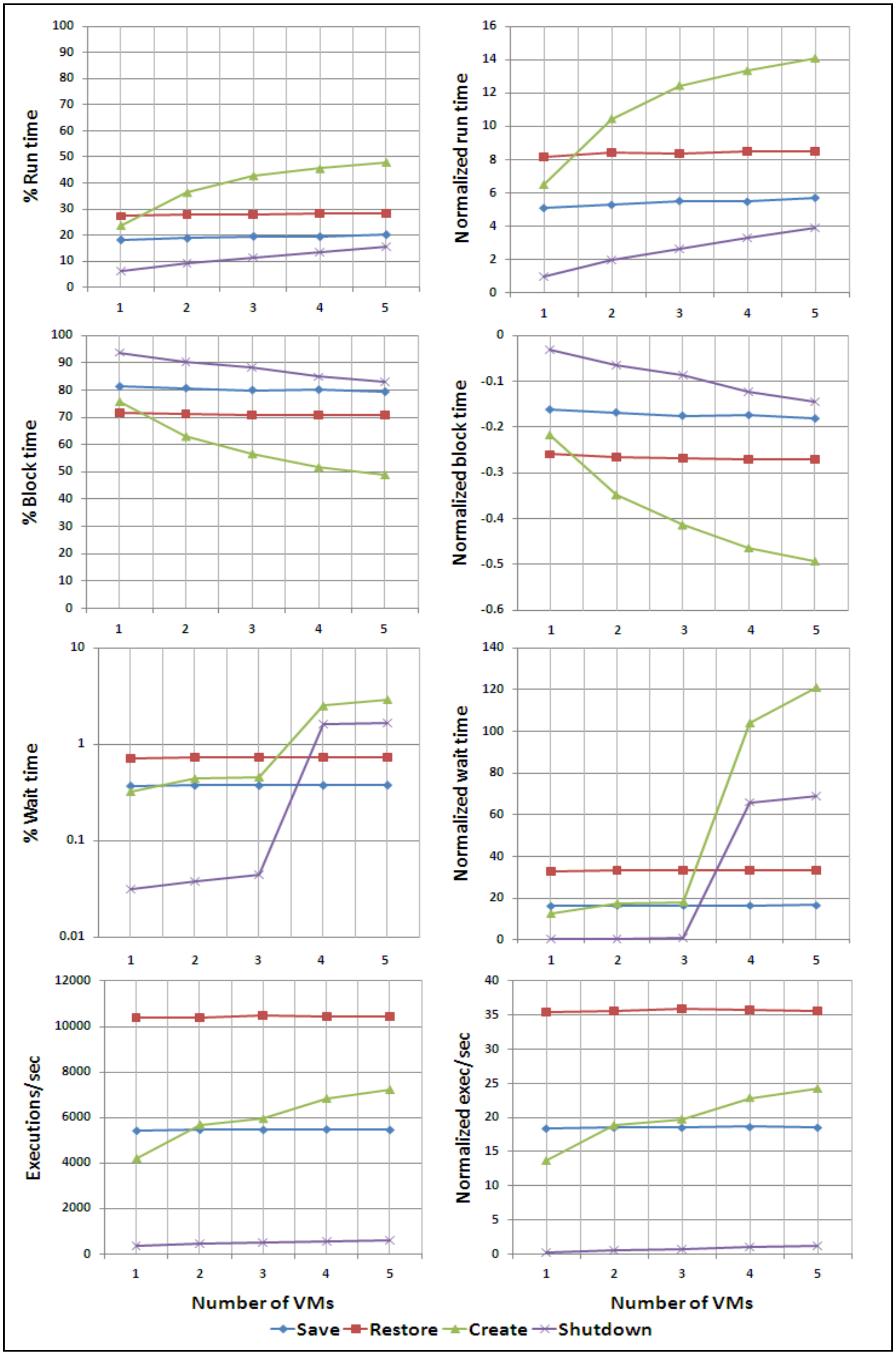


Figure 14: CPU Utilization for Create, Shutdown, Save & Restore operations

Finally, the time taken for each function applied on increasing number of VMs is plotted in Figure 16. Time taken for Save and Restore operations increases linearly with VM number due to increasing number of checkpoint files. In comparison, Creation time shows gradual rise while Shutdown time shows negligible effect.

.

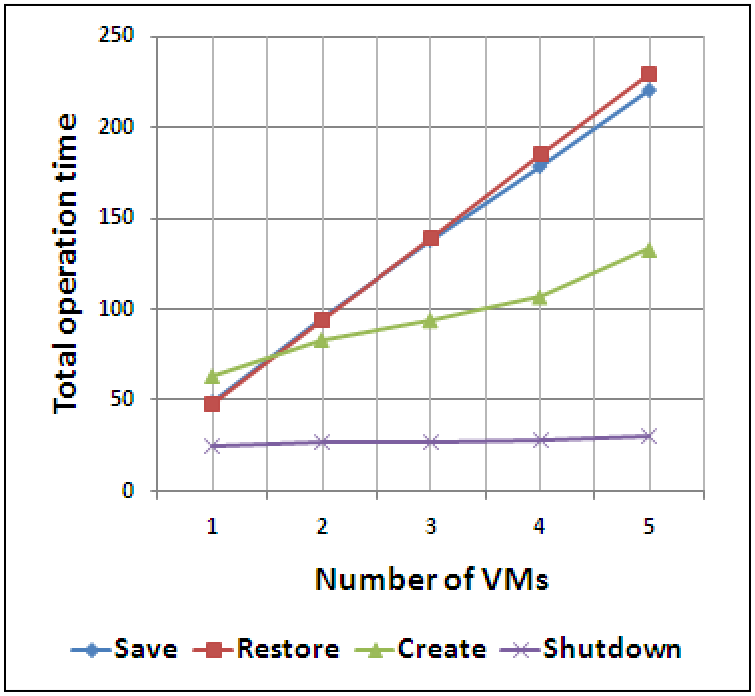


Figure 15: Effect of parallelization on operation time

The gradual rise of ‘Create’ time compared to ‘Restore’ time with increasing VM number shows that it would be preferable to create new machines against transferring idle ones from another machine for fixed memory size. Shutting down multiple virtual machines requires almost the same time irrespective of number of VMs, which makes a strong case for parallel machine shut down for maintenance or during overload.

6 SUMMARY, CONCLUSION AND FUTURE WORK

6.1 Summary

The central theme of this research was to make advances towards an autonomic virtual machine management framework. At the outset, we defined our goal of enabling a physical server to take decisions about managing its cluster of virtual machines without manual intervention and act intelligently to balance the workload in a data center for server consolidation. Our primary objective was to explore different approaches to help define a hypothetical model based on the pattern of VM interference and build an abstraction underneath Xen platform to effectively utilize the model's capabilities. We also attempted to leverage the functionality of the underlying architecture to expedite the process of virtual machine migration that forms the core issue in load balancing.

We began our analysis by examining the latest technology in backend data-centers, focusing our attention on the RDMA capability provided by the InfiniBand architecture. We integrated the DART implementation built over Portals C library with the Python base provided in Xen and concluded that modification to the Xen kernel was essential to allow InfiniBand adapters to provide virtual machine migration, bypassing the network layers using zero-copy mechanism. Further, we explored the utility based approach to VM management analogous to its foray in using task-dependant application management. Our goal was to synthesize the application utility on single VM w.r.t. various benchmarks, design a model of interference for multiple entrant virtual machines and elicit suitable actions based on the model's outputs. For the purpose of synthesis, we chose the ApacheBench and Httperf benchmarks for network and compute intensive tasks, isolation suite and FFTW providing compute and memory intensive tasks. To design the model, we studied the effect of fine granular tuning in CPU credit scheduler on application performance. Finally, we canvassed the effect of parallelization of Xen functions and disk image relocation on resource utilization so as to elicit the correct actions to be taken in case of system overload.

6.2 Conclusion

The management of a virtualized system can be a daunting task considering the plethora of control parameters available for tweaking, the complex interaction of multiple resources underneath and different user policies and QoS requirements dominating the decisions. There are limitations to the working of a system administrator whose primary purpose is to keep the systems running at whatever cost possible, compromising on performance. It is necessary to embed a model-based functionality in the hardware itself so as to reconfigure itself in case of unbalanced load or system problems.

Any system must equip its hardware for optimum performance before optimizing the software abstraction running on it. The use of an RDMA infrastructure underneath Xen platform would go a long way in speedy migration of virtual machines with negligible transfer overhead. Secondly, synthesizing a utility specific to an application as opposed to the virtual machine itself would enable physical machines to take application centric decisions favoring the customers and application developers alike. Xen provides excellent support for CPU re-configuration in terms of number of CPUs used, their absolute and relative weights in multiple VMs and sharing or pinning mechanisms. These can be used along with active monitoring of CPU utilization to elicit actions such involving manipulation of domains. Also, the concurrency of different Xen functions can be used to lower operation overhead and maximum resource utilization. The results in Chapter 5 shed light on these conclusions.

6.3 Future Work

Currently, the Xen kernel has no support for RDMA architecture or its abstraction. Specific modifications in the drivers can definitely aid in expedited virtual machine transfer.

We collaborated with researchers in Drexel University regarding elicitation and utilization of utility functions for self-assessment in an autonomic environment. We would like to go ahead with the synthesis to build a mathematical model to provide a map of good and bad states for a system to reside. Depending on the system state, remedial measures can be applied to move the system back to a stable state from an unbalanced state.

We did not account for the delayed effects resulting from the dynamic change in resource configuration, since we set the system parameters at inception. It would be interesting to see the behavior of an application with environment metrics changed on the run.

Our latter results mostly use a uniprocessor configuration running a single guest virtual machine. Multiprocessor and multiple guest analysis can be done to devise a more complex model of interference. We were unable to design a fully functional mathematical model taking all the mentioned control parameters in consideration and suggesting actions for specific VM behavior. This is definitely the next logical step in our research.

BIBLIOGRAPHY

A. Bertl, E. Gelenbe, M. D. Girolamo, G. Giuliani, H. D. Meer, M. Dang, and K. Pentikousis. "Energy-efficient cloud computing." *The Computer Journal*, 2009.

B. Sotomayor, K. Keahey, I. Foster. "Overhead Matters: A Model for Virtual Resource Management." *2nd International Workshop on Virtualization Technology in Distributed Computing*. 2006. p.5.

C. Docan, M. Parashar, S. Klasky. *Enabling High Speed Asynchronous Data Extraction and Transfer Using DART.* Rutgers University, NJ, 2007.

D. A. Menasce, M. N. Bennani. "Autonomic virtualized environments." *Intl. Conf. on Autonomic and Autonomous Systems.* 2006. 28.

D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, A. Kemper. "Energy-aware server provisioning and load dispatching fo rconnection-intensive internet services." *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC.* 2008. pp. 326-335.

D. Gupta, R. Gardner, L. Cherkasova. *XenMon: QoS Monitoring and Performance Profiling Tool.* HP Laboratories, Palo Alto, October 18, 2005.

H. Abdelsalam, K. Maly, R. Mukkamala, M. Zubair, and D. Kaminsky. "Towards energy efficient change management in a cloud computing environment." *Intl. Conf. on Autonomous Infrastructure, Management and Security.* 2009. 161-166.

HP Labs. *Httperf* . 2008. http://www.hpl.hp.com/research/linux/httperf/.

I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, R. J. Figueiredo. *VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing.* University of Florida, Gainesville, FL, 2004.

I. Kulkarni, M. Parashar, R. Muralidhar, H. Seshadri, S. Poole. *Investigating Application-Centric Aggressive Power Management for HPC Workloads.* 2009.

J. N. Matthews, E. M. Dow, T. Deshane, W. Hu, J. Bongio, P. F. Wilbur, B. Johnson. *Running Xen: A Hands-on guide to the Art of Virtualization.* Prentice Hall, 2008.

J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, J. Owens. *Quantifying the Performance Isolation Properties of Virtualization Systems.* Clarkson University, San Diego, CA, June 13-14, 2007.

J. O. Kephart, H. Chan, R. Das, D. W. Levine, G. Tesauro, F. Rawson, C. Lefurgy. "Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs." *Intl. Conf. on Autonomic Computing.* 2007. 24.

J. Rao, X. Bu, C. Xu, L. Wang, G. Yin. *VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration.* Wayne State University, MI, June 15-19, 2009.

J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. "Managing energy and server resources in hosting centers." *SIGOPS Oper. Syst. Rev., vol. 35, no. 5*, 2001: 103-116.

M. Frigo, S. G. Johnson. *FFTW.* 2005. http://www.fftw.org/.

M. Rosenblum, T. Garfinkel. "Virtual Machine Monitors: Current Technology and Future Trends." *The Computer Journal, vol. 38, no. 5*, 2005: 39-47.

M. Zhao, R. J. Figueiredo. *Experimental Study of Virtual Machine Migration in Support of Reservation of Cluster Resources.* University of Florida, 2007.

P. Apparao, R. Iyer, X. Zhang, D. Newell, and T. Adelmeyer. "Characterization and analysis of a server consolidation benchmark." *ACM SIGPLAN/SIGOPS international conference on Virtual execution environments.* 2008. pp. 21-30.

P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, A. Warfield. *Xen and the Art of Virtualization.* Cambridge, Oct 19-22, 2003.

P. Grandis, G. Valetto. *Elicitation and Utilization of Utility Functions for the Self-assessment of Autonomic Applications.* Drexel University, PA, 2009.

R. Das, J. O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, H. Chan. "Autonomic multi-agent management of power and performance in data centers." *Intl. joint Conf. on Autonomous agents and multiagent systems.* 2008. pp. 107-114.

R. Nathuji, C. Isci, and E. Gorbatov. "Exploiting platform heterogeneity for power efficient data centers." *Intl. Conf. on Autonomic Computing.* 2007. 5.

S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan. "VManage: Loosely coupled platform and virtualization management in data centers." *Intl. Conf. on Autonomic Computing.* 2009. 127-136.

S. Srikantaiah, A. Kansal, and F. Zhao. "Energy aware consolidation for cloud computing." *USENIX Workshop on Power Aware Computing.* 2008.

VMWare. "Understanding Full Virtualization, Paravirtualization and Hardware Assist." 2007.

W. Huang, Q. Gao, J. Liu, D. K. Panda. *High Performance Virtual Machine Migration with RDMA over Modern Interconnects.* Ohio State University, Columbus, Ohio, 2007.

W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation." *1st International Conference on Cloud Computing*. 2009. 254-265.

Y. Song, Y. Sun, H. Wang, and X. Song,. "An adaptive resource flowing scheme amongst vms in a vm-based utility computing." *IEEE Intl. Conf. on Computer and Information Technology*. 2007. pp. 1053-1057.