CERIAS Tech Report 2007-108
Adaptive Virtual Distributed Environments for Shared Cyberinfrastructures
by Ruth, Paul
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Graduate School ETD Form 9
(01/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis Acceptance

This is to certify that the thesis prepared

By Paul Michael Ruth

Entitled Adaptive Virtual Distributed Environments for Shared Cyberinfrastructures

Complies with University regulations and meets the standards of the Graduate School for originality and quality

For the degree of Doctor of Philosophy

Final examining committee members

Dongyan Xu, Chair

Daisuke Kihara

Bharat Bhargava

Mikhail Atallah

Vernon Rego

Approved by Major Professor(s): Dongyan Xu

Approved by Head of Graduate Program: Aditya Mathur/William J. Gorman

Date of Graduate Program Head's Approval: 19 July 2007

ADAPTIVE VIRTUAL DISTRIBUTED ENVIRONMENTS FOR SHARED CYBERINFRASTRUCTURES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Paul M. Ruth

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2007

Purdue University

West Lafayette, Indiana

UMI Number: 3291081

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

To my mother, *Jean Ruth*

ACKNOWLEDGMENTS

Throughout my experiences as a graduate student at Purdue, there are many people who have influenced my development as a researcher and some who have contribute directly to the success of my dissertation. Several should be thanked here:

- Dongyan Xu, my advisor who has supported me throughout my research and has put up with my, sometime, slow progress. Without Dongyan I would not have the experience needed to successfully navigate academia.

- Committee Members, Mikhail Atallah, Bharat Bhargava, Daisuke Kihara, and Vernon Rego, for their guidance throughout the process and there time and effort spent on my committee.

- Everyone in the lab: Xuxian Jiang, Junghwan Rhee, Ardalan Kangarlou, Ryan D. Riley, and Zhiqiang Lin. Without the engaging discussions none of these ideas would be as developed. Special thank to Xuxian for initiating the VIOLIN project, as well as, Junghwan and Ardalan for their continued work on it.

- The nanoHUB crew.: Sebastien Goasguen, Gerhard Kilmech, Rick Kennell, and Steve Clark. Their hard work in creating, promoting, and funding the nanoHUB has contributed to my work more significantly than they could ever know.

- Ron Minnich, my internship advisor at LANL,for convincing me to keep going when I didn't have will. Without him I most certainly would not be where I am today.

- Dennis Brylow, who continues to demonstrate that it is possible to get what I want out of academia and has supported me throughout it all.

- Adam Welc, for many memorable nights at the LBC. Na zdrowie.

- Debbie Hughes who always knew I could do it and who has contributed to my dissertation by putting up with years of my whining and has helped improve my writing. She has read this dissertation at least as many times as I have. For her reward, she gets a lifetime of putting up with my whining and helping me to improve my writing skills.

- The National Science Foundation for suppporting my work through grants OCI-0438246, OCI-0504261, and CNS-0546173.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

API	Application Programming Interface
BLAS	Basic Linear Algerbra Sub-Routines
CI	Cyberinfrastructure
IP	Internet Protocol
ISO	International Organization for Standardization
JVM	Java Virtual Machine
HPC	High Performance Computing
NIC	Network Interface Card
NSF	National Science Foundation
MPI	Message Passing Interface
P2P	Peer-to-Peer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	User-Mode Linux
VMM	Virtual Machine Monitor
VE	Virtual Environment
VIOLIN	Virtual Internetworking on OverLay INfrastructure
VM	Virtual Machine

ABSTRACT

Ruth, Paul M. Ph.D., Purdue University, August, 2007. Adaptive Virtual Distributed Environments for Shared Cyberinfrastructures. Major Professor: Dongyan Xu.

A shared distributed cyberinfrastructure is formed by federating computation resources from multiple domains. Such shared infrastructures are increasing in popularity and are providing massive amounts of aggregated computation resources to a large numbers of users. Meanwhile, virtualization technologies, at machine and network levels, are maturing and enabling mutually isolated virtual computation environments for executing unmodified parallel and distributed applications on top of such a shared physical cyberinfrastructure.

In this dissertation, we go one step further by supporting runtime adaptation of virtual computation environments as integrated, active entities. More specifically, driven by both dynamic availability of infrastructure resources and dynamic application resource demand, a virtual computation environment is able to automatically relocate itself across the infrastructure and adjust its share of infrastructural resources. Such runtime adaptation is transparent to both users of a virtual environment and applications running inside the environment, providing the look and feel of a private, well-provisioned computational environment to the users and the applications.

This dissertation presents the design, implementation, and evaluation of a middleware system, VIOLIN, which enables adaptive virtual computational environments. Each VIOLIN virtual environment is composed of a virtual network of virtual machines and is capable of live migration across a multi-domain physical infrastructure. This research effort includes the development of the enabling mechanisms for virtual environment creation and adaptation, as well as, policies for effectively utilizing these mechanisms. Through this effort, we have found that by combining virtual environ-

ments with adaptation policies and mechanisms, we can provide significant increases in both usability and performance of shared computational infrastructures. Further, the VIOLIN middleware is currently deployed on a production cyberinfrastructure, called the nanoHUB, as a major computational facility used to execute parallel and distributed nanotechnology simulations.

1. INTRODUCTION

1.1 Background and Problem Statement

The growth of distributed infrastructures such as the Grid [1, 2] and PlanetLab [3–5], along with the current push toward shared cyberinfrastructure has made computing and communication resources available to a large community of users. Meanwhile, virtualization technologies [6–9] are increasingly being deployed on top of such shared physical infrastructures [10–14], supporting the customization, mutual isolation, and administrator privilege of virtual machines (VMs) running many applications on behalf of users with a diverse set of needs. With recent advances in network virtualization techniques [15–18], virtual private networked environments can also be created on top of a shared distributed infrastructure.

In this dissertation, we argue that machine, network, and, ultimately, environment virtualization provide a number of key features needed to federate distributed shared resources into isolated, dynamic, and customizable platforms for a wide array of computing needs. Previous approaches to infrastructure sharing do not use virtualization, limiting the types of applications supported and requiring significant and often infeasible administrative and developer support to be usable. Further, the adaptive capabilities of virtualization create possibilities which exceed the current expectations of cyberinfrastructure. While an application is running, the amount, type, and location of resources allocated to the application can be modified. For example, an application that enters a phase of high CPU utilization can be dynamically allocated a larger share of the local CPU. Further, if the local CPU does not have enough available power, the application can be migrated to a host with adequate available CPU.

As such, this dissertation proposes an integrated virtualization-based framework for the federation of resources in a shared cyberinfrastructure. At the center of this framework is the VIOLIN middleware which enables virtual distributed environments [15, 19, 20] consisting of virtual machines connected by virtual networks. The key benefit of VIOLIN is that it decouples the ownership, configuration, administration, and resources of the virtual environments from those of the underlying infrastructure. Within a VIOLIN virtual environment, the user is able to execute and interact with unmodified applications, including parallel and distributed applications. Any negative impacts of these applications will be contained within the virtual environment and will not cause damage to the infrastructure.

The all-software virtualization of VIOLIN decouples the virtual environments from the underlying resources. This level of indirection provides a new opportunity to make these environments autonomically adapt to environmental and application dynamics [21–23]. The overall vision is to enable virtual environments that cater to the application running inside them. The virtual environments dynamically adapt and re-locate themselves by scaling local resource allocation and migrating across the multi-domain infrastructure. From the point-of-view of a distributed application, the underlying virtual environment may change its CPU, memory, and network link capacity, re-configure its topology and scale (e.g., by adding new virtual machines), or even migrate its virtual machines to more resource-sufficient locations in the infrastructure. Conversely, from the point-of-view of the underlying infrastructure, scaling and migration of virtual environments allows for more efficient utilization of available resources. Virtual environments that are using less than their allocation of resources can have their allocation scaled back, freeing resources to be used by other under-provisioned virtual environments. Virtual environment adaptation is a unique capability that cannot be achieved by a physical environment which is restricted by the limits of physical computing with static resource capacity. Complementing the traditional adaptation scheme where *applications adapt to dynamic environments*, we advocate a new scheme where (*virtual*) *environments adapt to dynamic applications.*

To realize the vision of autonomic adaptive virtual computation environments in a physical distributed infrastructure that spans multiple domains, we address the following challenges:

The first challenge is to enable virtual environments that address the needs of computational applications while spanning domains. The necessary characteristics of these virtual environments are that they can achieve near-native performance while executing unmodified existing applications and remain isolated from their underlying host domains. Existing virtual machine platforms can execute unmodified applications while providing good performance and per host isolation. However, virtual machines do not necessarily create virtual environments. In order to create virtual environments, VIOLIN must isolate and connect groups of distributed virtual machines into what appear to be private LANs. These private LANs are called virtual environments and can be used to run any parallel or distributed applications.

The second challenge is to provide the mechanisms for application-transparent virtual environment adaptation. In order to provide a stable environment, adaptation should occur transparently without effecting the application or the user. Work has been done to enable resource reallocation and migration within a local-area network [24,25] and machine migration features are provided by the most current machine virtualization platforms. One question still demands our attention: how can we migrate an entire virtual environment across a wide-area network without effecting the application? The solution must keep the virtual machines alive throughout the migration. Computation must continue and network connections must remain open. The necessary wide-area migration facility requires two feature not yet provided by current virtualization techniques. First, virtual machines need to retain their IP addresses and remain accessible through the network despite physical routers not knowing that they were migrated. Second, wide-area migration cannot rely on NFS to maintain a consistent view of the large virtual machine image files. These files must be quickly transferred across the relatively slow wide-area network. Current solutions are not adequate for wide-area use.

The third challenge is to define *allocation policies*: The static allocation of shared resources considers the available resources and requested resources to find the optimal allocation. Autonomic environments, however, must have the intelligence to scale resource allocations without user intervention. How do we know when a virtual machine needs more CPU? Which virtual machine should be migrated if a host can no longer support the memory demands of its guests virtual machines? If a virtual machine needs to be migrated, where should it go? We must determine whether the best destination is the one which affords the quickest migration, or privilege the one which results in more adequate resource allocation.

The VIOLIN framework provides solutions to the above challenges. Based on the VIOLIN framework, we have deployed a production system on the nanoHUB [26] infrastructure, as well as, created experimental prototypes to evaluate the effectiveness of adaptation mechanisms and policies on increasing the usability and utilization of shared computational resources. Further, there are over 20,000 users of the nanoHUB, hundreds of which have accessed nanotechnology simulations [27] that run on production computational clusters that are actually VIOLIN virtual environments. These users have run thousands of parallel and distributed jobs during the first year of VIOLIN's deployment. As new components of the experimental prototype become reliable they will be deployed in the production system for the benefit of nanoHUB users.

1.2 Dissertation Contributions

The challenges and contributions of this dissertation are three-fold: the development of the VIOLIN middleware which enables virtual environments, the development and evaluation of virtual environments for inter-domain sharing, and the development and evaluation of adaptive virtual environments for shared cyberinfrastructures.

- **VIOLIN middleware**. As the basis of this dissertation the VIOLIN middleware was designed, implemented, and evaluated for effective utilization of

shared cyberinfrastructures. VIOLIN's key contributions is on-demand creation of isolated, dynamic, secure, and customizable virtual environments that can effectively utilize the distributed resources of the shared cyberinfrastructure. This is accomplished by inserting VIOLIN as a layer of indirection sitting between the infrastructure and virtual machines running on top of it. This layer provides users with the familiar look-and-feel of a private LAN environment while allowing the cyberinfrastructure to be shared. Effectively and efficiently sharing the infrastructure is achieved by manipulating the scale and location of each virtual environment's resource allocation. VIOLIN is among the first virtual environment middleware that allows live wide-area migration, and it is one of the few being used in a production environment.

- **Virtual environments for inter-domain sharing**. Leveraging the abilities of VIOLIN, we have designed, implemented and evaluated a system of virtual environments for inter-domain sharing. VioCluster [28] is a system that allows multiple independently administered computational clusters to share resources by temporarily lending and borrowing cluster nodes. We observe that many computational clusters go through phases of utilization. They often sit idle for long stretches of time followed by periods of demand beyond their capabilities. The VioCluster system monitors the resource demands of each cluster by communicating directly with their batch schedulers. The number, size, and runtime of the jobs in the scheduler are applied to domain-specified policies in order to facilitate the lending of idle nodes of some clusters to others that are overloaded. The system has been shown to mitigate large spikes in cluster load. Existing virtualization-based cluster load-balancing mechanisms can only be used to balance load within a single physical cluster. VioCluster can safely distribute load beyond domain boundaries while maintaining strict autonomy of each administrative domain.

- **Adaptive virtual environments for shared cyberinfrastructure**. We have developed, implemented, and evaluated a system of virtual environments based on VIOLIN intended to improve application performance and increase the efficient use of a shared cyberinfrastructure. This system enables the mechanisms and policies for virtual environment adaptation. Users and user groups utilize cyberinfrastructure resources through isolated VIOLIN virtual environments customized for their specific needs. Further, each environment transparently relocates itself through the cyberinfrastructure consuming resources and adapting to the environmental characteristics of its internal applications. In contrast to VioCluster, adaptation decisions are made using only data observable from outside of the virtual environments (e.g. CPU, memory, and network utilization). The system predicts the resource needs of applications running within the virtual environments without any knowledge of the specific application. The predictions are used to determine the appropriate way in which to adapt resource allocation to the needs of applications running within the virtual environments. The ability to transparently adapt environments to arbitrary applications is essential to the efficient use of cyberinfrastructure. Our study shows that these techniques greatly increase application performance and overall infrastructure resource utilization.

1.3 Dissertation Organization

This dissertation is organized into 7 chapters including this introduction chapter. Chapter 2 lays out the framework for the dissertation. Chapter 3 discusses related works. Chapter 4 presents the details of VIOLIN enabling mechanisms. Chapter 5 presents the application of VIOLIN to inter-domain cluster sharing. Chapter 6 discusses adaptation mechanisms and policies for VIOLIN virtual environments deployed on a shared cyberinfrastructure. Concluding remarks and future work are discussed in Chapter 7.

2. OVERVIEW OF VIOLIN FRAMEWORK

This chapter presents an overview of the VIOLIN integrated framework for utilizing cyberinfrastructure.

2.1 Overview

Figure 2.1 shows the overall organization of the VIOLIN framework. There are three main components whose development and concepts build upon each other to create fully adaptive virtual environments for cyberinfrastructure: (1) the basic virtual environment enabling middleware (VIOLIN), (2) adaptive inter-domain sharing

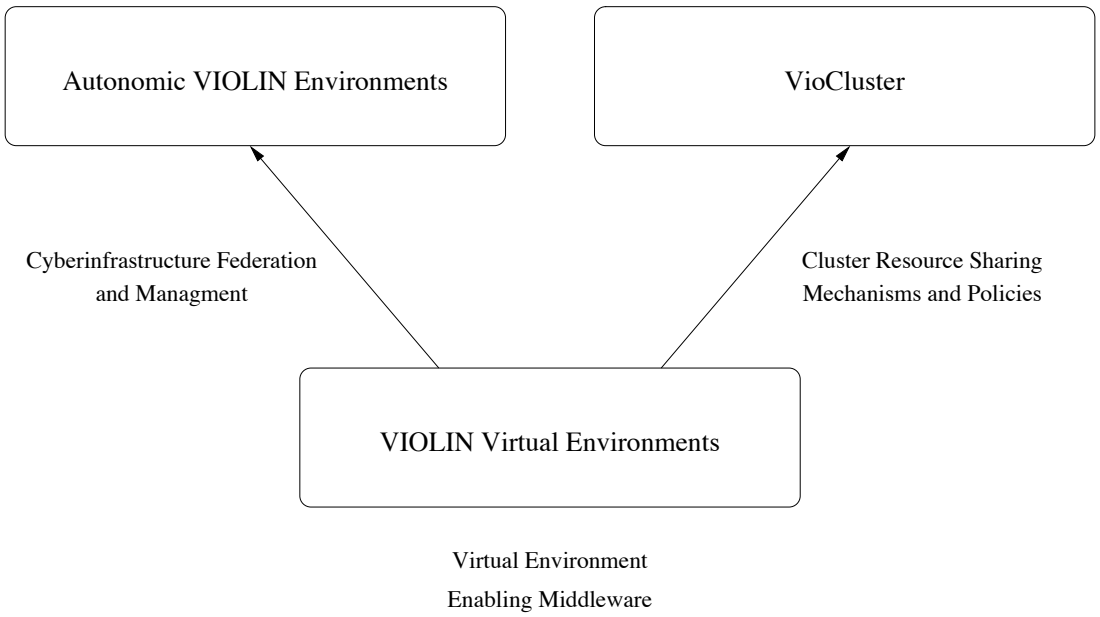


Fig. 2.1. Road map toward the integrated VIOLIN framework for virtual environments in a shared cyberinfrastructure

mechanisms and policies (VioCluster), and (3) mechanisms and policies for autonomic adaptive virtual environments in shared infrastructures.

2.1.1 Underlying Cyberinfrastructure

Below the adaptive virtual environment framework described in this thesis sits the cyberinfrastructure. There is growing realization that computers, networks, and network-attached devices have developed to the point where any device, regardless of location, can be used by any scientist anywhere in the world to perform tasks not possible using local infrastructure. Cyberinfrastructure is the term created by the National Science Foundation (NSF) to describe its vision of the federation of a large number of distributed resources to be used by a wide range of users. The challenges to the development of a cyberinfrastructure are (1) facilitation of cooperation between independently administered devices and (2) the need to provide diverse users and user groups with a simple, easy to use interface through which to utilize resources that are heterogeneous, distributed, and independently administered.

In 2003, the NSF released a blue-ribbon advisory panel report on cyberinfrastructure [29]. In their report, the NSF presented its desire to create shared cyberinfrastructures that enable science and engineering innovation. The work presented in this dissertation is among work that shares the long-term vision.

2.1.2 VIOLIN Virtual Environments

The goal of the next generation cyberinfrastructure is to federate massive amounts of heterogeneous resources that are available through the Internet and make the resources available to users. Unfortunately, existing methods for sharing independently administered resources have limitations. Theses methods often use restrictive user authentication models that require manual creation and monitoring of accounts within each independent domain. Independent authentication in each domain hinders the ability to federate infrastructures. In addition, the heterogeneity seen across do-

mains limits the portability of most applications. In order to provide for portability, some cyberinfrastructure projects [30–32] require application programmers to use infrastructure specific API's and link their codes with infrastructure specific libraries. Most users, especially those who are not computer experts, would prefer to remain unencumbered by the details surrounding access to resources and do not wish to, or may not be able to, modify or recompile existing codes. We argue that the virtualization of cyberinfrastructure resources, through both virtual machines and virtual networks, can allow them to be used as if they were resources configured and administered locally.

The VIOLIN middleware interacts with the infrastructure to form the foundation of our integrated framework. The VIOLIN middleware manages the creation, destruction, and adaptation of multiple virtual environments sharing the resources of the infrastructure. Each virtual environment is composed of one or more virtual machines connected through an isolated virtual network. The VIOLIN middleware uses machine and network virtualization to decouple the underlying infrastructure from the virtual environment and provide cyberinfrastructure users with a familiar look-and-feel of a customized private local-area network. Users can execute unmodified applications (both new and legacy) as if they were utilizing dedicated local resources. Meanwhile, the cyberinfrastructure is relieved of its per-domain user account maintenance responsibilities. Domains can participate in the cyberinfrastructure by providing support of virtual machines and agreeing to host virtual machines from the participating domains.

Although enabling VIOLIN virtual environments is a major contribution in itself, the adaptation abilities of virtual environments provide a unique opportunity. VIOLIN provides fine-grain control of the amount of resources (CPU, memory, and network bandwidth) allocated to each virtual machine within a VIOLIN environment. Further, it provides coarse-grain control by enabling the live migration of individual virtual machines, or whole virtual environments, between physical hosts or domains.

In 2005 VIOLIN was deployed on a production cyberinfrastructure called the nanoHUB [26]. The nanoHUB is a web-based system providing access to high-performance nanotechnology simulations. Since VIOLIN's deployment it has supported the execution of thousands of simulations for hundreds of users and has become an essential part of the nanoHUB's computing backend. As of this writing, production VIOLIN environments are utilized to run many of the most common parallel and distributed nanoHUB applications and have become some of its most secure, stable, and reliable computational facilities. VIOLIN virtual environment enabling mechanisms are presented in Chapter 4.

2.1.3 Adaptive Inter-domain Cluster Sharing

With VIOLIN virtual environments in place, we have created an integrated autonomic adaptation platform that monitors and adapts virtual environments to the applications they are running. By placing autonomic adaptation control facilities on top of VIOLIN environments and adapting the allocation of resources to applications we gain unique advantages in investigating efficient utilization of computational infrastructure.

The VioCluster adaptive inter-domain cluster sharing system is the first system built upon the VIOLIN middleware. VioCluster allows independently administered computational clusters to share resources while each cluster retains isolated administrative authority. VioCluster leverages VIOLIN's ability to virtually extend a physical cluster to a remote domain through the on-demand creation of virtual machines as part of the physical cluster. VioCluster's most significant contributions are to adaptation policies implemented to maximize the performance benefits of adaption through defining effective cluster-specific node lending and borrowing policies. The policies increase the overall performance of the system while protecting individual clusters from the negative effects of lending nodes.

Chapter 5 presents the implementation of the VioCluster system and the study of its ability to increase the performance of computational clusters.

2.1.4 Autonomic Virtual Environments for Shared Cyberinfrastructure

VIOLIN and VioCluster together demonstrate the benefits of adaptive virtual environments. VioCluster, however, does not use all of the dynamic abilities of VIOLIN. It only uses on-demand creation, destruction, and expansion of virtual environments. Building on the adaptive concepts applied by VioCluster, we created autonomic virtual environments for wide-area multi-domain cyberinfrastructure that can scale resource allocation and migrate live virtual environments across distributed infrastructure. These fully autonomic virtual environments have no dedicated hosts, instead, they exist in the dynamic cyberinfrastructure and flow through the available resources consuming resources and adapting to environmental changes. The primary contribution of autonomic virtual environments is to the creation of adaptation mechanisms and algorithms.

Chapter 6 presents the implementation of the autonomic virtual environment system and discusses the experimental deployment of the system on the nanoHUB's infrastructure.

3. RELATED WORK

3.1 Virtual Machines

Virtual machines have been used for computing since the early 1960's when they were developed along with other forms of time-sharing operating systems [33]. Before the introduction of virtualization, computing systems research followed the paradigm where in one program executed on one machine at any given time. Time-sharing operating systems changed the way computers were used by allowing multiple programs to run on a single computer, seemingly at the same time. At their inception, there were two purposes for developing and using time-sharing systems, (1) to more efficiently use available computing resources, and (2) to create a more productive interface for humans to interact with computers [34]. To this day, the motivations behind modern virtualization platforms remain the same as over 40 years ago. Advances in virtualization continue to increase the efficiency of computational capacity as well as provide convenient interfaces to massive amounts of computational power.

The earliest notable virtual machines were developed as a time-sharing system for IBM's System/370. Often simply called VM or VM/370, Virtual Machine Facility/370 [35] is a second generation time-sharing system that pioneered the use of virtualization. VM/370 allows multiple users to simultaneously interact with their own copy of a single-user operating system called Conversational Monitor System (CMS). Collectively, all CMS instances simultaneously use a single processor and are controlled by the Control Program (CP). The CP multiplexes the available resources while maintaining strict isolation between each operating system. Remarkably, the CP/CMS model remains very similar to modern virtualization methods. The modern equivalents of the Control Program are called Virtual Machine Monitors (VMM) or

Hypervisors and virtual machines now commonly run off-the-shelf multi-user operating systems.

Throughout the 1960's and 1970's considerable research went into developing virtual machines as time-sharing mechanisms. However, from the 1970's to the mid-1990's operating systems such as UNIX, Linux, BSD, Solaris, Windows and many others that use the more familiar single kernel multiple-user paradigm became very popular while interest in virtualization has diminished.

More recently, machine and network virtualization has seen a renaissance. It has become common to use personal computers simultaneously running multiple operating systems that free the user from the limitations imposed by any individual operating system. Further, using powerful servers, it is now possible for large server farms to be completely based on virtual machines for easy management and flexible resource provisioning. Following the philosophy of the original virtual machines, modern virtual servers efficiently share the limited resources of a server farm while maintaining isolated customized servers for their customers. In addition, virtual servers can be moved between hosts within the server farm to avoid maintenance downtime and increase performance and availability.

3.1.1 Hardware Level Virtualization

The original virtual machines used in VM/370 were what today are called hardware level virtual machines. Hardware level virtualization is distinguished from other types of virtualization by the technique of creating virtual hardware that exactly replicates real hardware. The unmodified guest operating system interacts with familiar virtual hardware. In the case of VM/370 virtual machines, each copy of the CMS operating system interacts with its own virtual IBM System/370.

By providing replica virtual hardware, hardware level virtualization has the crucial benefit of supporting a wide variety of off-the-shelf operating systems. Platforms like VMware Server can easily support multiple virtual machines running unmodified

versions of Windows, Linux, Solaris, BSD, or most other operating systems for x86 architectures. The flexibility and ease of use of these systems continues to drives their popularity.

Virtualizing raw hardware, however, incorporates unnecessary overhead. Much of the I/O and networking capabilities of virtual machines can be made significantly more efficient by optimizing the virtual hardware. For this reason, many hardware level virtualization platforms use a method called *paravirtualization*. In paravirtualization, the strict separation between the VMM and the guest OS is broken. The VMM (often referred to as a *hypervisor* in paravirtualization) provides a modified set of virtual hardware that interacts with a modified guest OS. Xen is one of the more popular examples of paravirtualization. The paravirtualization optimizations in Xen allow it to provide higher performance than VMware Server, however, commercial operating systems, for example all versions of Microsoft Windows, cannot be run on Xen without special support from the manufacturer. It is important to note that hardware support for virtualization seen in the Intel Virtualization Technology (VT) chips [36] and AMD Pacifica [37] has become more common and allows for unmodified guest operating systems to run using some paravirtualization techniques while achieving better performance [38].

The following is a list of many of the popular hardware virtualization platforms and a brief description of the unique properties of each:

- **VMware** [39]. Possibly the most popular virtualization platform for business and personal use, VMware has a wide range of virtualization products including, VMware Workstation, VMware Player, VMware Server (formerly VMware GSX Server), and VMware ESX Server. VMware Server is a free version of VMware that runs on top of an existing operating system (currently Windows, Linux, and OS X). It supports any x86 operating system as a guest but suffers from reduced I/O performance due to it pure machine level virtualization. VMware ESX Server [40, 41] is not free but is a high performance paravirtualization

system with increased I/O performance [42]. VMware ESX Server also includes live migration capabilities [25].

- **Xen** [6]. Xen is the most popular open-source paravirtualization platform available today. Possibly the most popular virtualization platform in academia, Xen has a large community of active developers and is easy to modify. Xen provides both good performance [43] and live migration capability [24].

- **User-Mode Linux** [7]. Originally developed as a platform to simplify Linux kernel development, UML was never intended to be a full-fledged virtual machine platform. However, its fully user-level implementation and open-source Linux-based implementation makes UML a ideal candidate for many uses.

- **Parallels** [44]. Originally a competitor of VMware, Parallels became popular with the release of the Parallels Desktop for Mac. At the time, VMware Server did not support Mac OS X hosts. Many OS X users run Windows as a guest on Apple hardware using Parallels.

- **Microsoft Virtual PC** [45]. Virtual PC has been around since 1997. Originally, it was designed to support Windows OS's running on Power-PC Macintoshs. When virtualization began its renaissance, Microsoft bought Virtual PC and continues to develop it as a virtualization platform for Windows.

- **Kernel Virtual Machine (KVM)** [46] . KVM is a loadable kernel module for Linux based on QEMU [47]. KVM provides full virtualization from within the Linux kernel and has been included as part of the mainline Linux kernel since version 2.6.20. Although it is in its infancy, initial evaluation shows that KVM may provide better performance than Xen [48]

3.1.2 Operating System Level Virtualization

Operating system level virtualization is another popular form of virtualization that is implemented at a different level of the software stack. What differentiates

operating system level virtualization from hardware level virtualization is that there is no true VMM, instead there is a single instance of an operating system that is shared by all virtual machines. Virtual machines are isolated from one another through a kernel level abstraction but they share the same process scheduler, network stack, and device drivers. Because there is only one operating system, these systems are light weight and have higher performance than machine level virtualization systems. However, they are less customizable and may not provide as high a level of isolation (i.e. virtual machines often share a single IP address and set of ports).

Popular operating system level virtualization platforms include:

- **Linux-VServer** [49]. VServer is an open-source Linux-based OS level virtualization platform. It is mature and stable enough to be included with many popular Linux distributions and is the fundamental virtual abstraction used by PlanetLab [4].
- **Virtuozzo [50]/OpenVZ [51]**. OpenVZ is very similar to Linux-VServer. Most notably, Virtuozzo is based on OpenVZ. OpenVZ is one of the few operating system level virtual machines systems that provides for VM migration [52].
- **FreeBSD Jails** [53]. Operating system level virtualization for BSD.
- **Solaris Containers** [54]. Operating system level virtualization for Solaris.

3.1.3 Virtualization and Dynamic Adaptation

Beyond isolation and customization, the most useful but least understood aspects of virtualization are its dynamic adaptation capabilities. Much work has been done in creating virtual machines that perform nearly as well as raw hardware [55, 56]. However, we have just scratched the surface on virtualization's ability to provide better performance through dynamic application-transparent adaptation.

Resource Scaling

Many virtualization platforms provide facilities for controlling the resource allocation of virtual machines at runtime. Most commonly, the VMM can scale the amount of CPU capacity, memory, and network bandwidth allocated to a virtual machine.

Xen 3.0, for example, uses Slacked Earliest Deadline First (sEDF) [57] processor scheduling which provides weighted CPU sharing among virtual machines. Each virtual machine is allocated a percentage of the CPU and the Xen hypervisor guarantees the allocations. Further, the allocated percentage of CPU can be changed at any time during the life of the virtual machine.

Xen also allows for the scaling of memory allocation. While a virtual machine is running, the amount of memory a virtual machine is using can be increased or decreased up to a maximum determined upon the creation of the virtual machine. This feature is quite impressive when the effect on the guest operating system is considered. To enable memory scaling, the guest operating system needs to be able to handle changes in memory size at runtime. Most operating systems have not been implemented with this feature. Clearly, Xen's ability to modify memory size at runtime does not work with unmodified guest operating systems and is a feature of paravirtualization. Pure machine based virtualization methods (including VMware Server) using unmodified guest OS's can not modify memory allocation at runtime.

Machine Migration

Among the most sophisticated features of any virtual machine platform is the ability to migrate virtual machines between hosts. Virtual machine migration involves pausing a VM, transferring its state to another host, and restarting it. The challenge to migrating virtual machines is maintaining access to the state of the virtual machine and maintaining an appropriate network configuration, both of which are necessary in order to reinstantiate the virtual machine. In current virtualization platforms, including Xen and VMware ESX Server, these challenges are met by restricting mi-

gration to within a single LAN or layer-2 network. The virtual machine's filesystems are stored in NFS accessible storage. Both the source and destination hosts must have access to the NFS store eliminating the need to transfer the larger filesystem images. Further, networking configurations can remain unchanged due to the bus architecture used by many common layer-2 networks, including Ethernet. Before and after the migration, the virtual machine remains part of the local Ethernet and can send and receive data without modifying the network configuration. In contrast, VIOLIN virtual environments can migrate live across a wide-area network due to the implementation of virtual networking at a lower level in the network stack (layer-2).

Some platforms, including Xen and VMware ESX Server, allow for the live migration of virtual machines. In live migration, the virtual machine remains running through the execution of the migration. When performed efficiently, the migration takes approximately as long as is needed to transfer the virtual machine's memory from the source host to the destination. However, the actual downtime the virtual machine experiences is less than one second [24]. These live migration facilities are key to VIOLIN's ability to provide wide-area adaptation of virtual environments.

3.2 Virtual Networks

There are several virtual networking systems which are similar to VIOLIN networking. Each of these systems has its own benefits and limitations.

- **VNET**[1] [17,58]. The VNET project at Northwestern University creates virtual network overlays composed of virtual machines residing on distributed hosts. VNET provides a virtual a layer-2 Ethernet that connects remote virtual machines to a local physical LAN. As opposed to VIOLIN, VNET does not maintain strict isolation between virtual networks. Instead, VNET creates isolation on a shared LAN by assigning IP addresses to virtual machines such that the

[1]Several similar projects share the name 'vnet'. We discuss Peter Dinda's VNET project from Northwestern University

co-located virtual networks use disjoint sets of IP addresses. In this configuration, misconfigured IP virtual machines can have negative effects on other virtual networks. One salient feature of VNET is the ability to automatically find shorter paths between virtual machines in the virtual network through Internet increasing performance of applications running inside the virtual network [18, 59].

- **IPOP and ViNE**. IPOP [60] and ViNE [61, 62] are both projects created at the University of Florida's Advanced Computing and Information System Laboratory. IPOP and VIOLIN share the P2P layer-2 approach to virtual networking. A P2P network is utilized to tunnel Ethernet frames between virtual machines. ViNE virtualizes at the IP layer (layer-3) and is, therefore, similar to traditional corporate VPN's. ViNE is limited, however, by requiring the modification the routing tables of the local IP level Internet edge routers.

3.3 Virtual Clusters and Workspaces

A system which shares similar goals to VIOLIN is Cluster-On-Demand (COD) [21]. Although Cluster-On-Demand does not use virtual machines, it allows dynamic sharing of resources between multiple clusters based on the same pool of hosts. In a similar fashion to Oceano [63] and Emulab [64], Cluster-On-Demand reallocates resources by using remote-boot technologies to re-image a physical machine and install preconfigured disk images from the network. The disk image that is installed determines which cluster the nodes will belong to upon booting. In this way, Cluster-On-Demand can redistribute the resources (at the granularity of a node) of a cluster among several logical clusters sharing those resources. When we compare our work with Cluster-On-Demand we see two projects that have very similar goals; however, Cluster-On-Demand works by reinstalling the base operating system of the machines. Our work creates virtual environments running on top of the existing VMMs and is designed to be deployed over wide-area networks. Cluster-On-Demand is more

suited for a centralized pool of machines supporting logical clusters administered by trusted local administrators. The authors of Cluster-On-Demand are also studying market-based algorithms for negotiating control over resources in a shared cluster environment [65, 66]. We are also interested in machine broker policies, however, we do not focus on market-based strategies with VIOLIN.

More recently, the group working on Cluster-on-demand has developed a new project called Shirako [67–71]. Shirako is a virtualization-based cluster system in which multiple virtual clusters are deployed on a single host cluster. Similar to our work, the distribution of resources to virtual clusters changes as demand for cluster resources changes. Shirako is limited to a single host cluster and does not extend beyond domain boundaries.

The Virtual Workspaces [12, 13, 72, 73] project at Argonne National Laboratory has the goal of providing policy-driven resource management and dynamically deployable environments for Grid applications. As part of the Globus project, Virtual Workspaces has the potential to be integrated with a widely deployed Grid meta-scheduler. Up to now, the work on Virtual Workspaces has focused on just-in-time deployment of virtual machine through Grid meta-schedulers [74].

PlanetLab [3] is a infrastructure consisting of computers distributed across the World and made available for experiments on networking and distributed systems. Users can acquire virtual machines, called 'slices', on any of the cooperating hosts. PlanetLab is intended to be a test bed for distributed systems that possesses the characteristics of real Internet overheads and congestion. Further, PlanetLab does not create isolated virtual networks. It is only intended as a platform for world-wide distributed system experiments. A limitation of PlanetLab is its use of VServer. Using VServer, all virtual machines on a host share a single IP address and must cooperate when accessing network ports.

3.4 Grid Computing and Cyberinfrastructure

Currently, the most common computational resource sharing methods are seen in the creation of large, shared Beowulf [75] style computer clusters that multiplex resources through a batch scheduler such as PBS [76] or Torque [77]. A common example of such systems would be a large general-purpose cluster administered by a campus-wide authority in a university system and used by members of many departments. More recent examples of decentralized resource sharing include cycle stealing applications such as SETI@Home [78], as well as meta-scheduling dedicated Grid infrastructures like Globus [1, 79], Condor [30], and Sun Grid Engine [80]. All of these solutions provide access to seemingly endless amounts of computational power without incurring the full cost of ownership. However, common to all of these systems is the problem that jobs are run on nodes over which the job owner has no control.

Another interesting system that uses virtual machines for resource sharing is In-VIGO [10, 81–84]. In-VIGO is a distributed Grid environment supporting multiple applications which share resource pools. The In-VIGO resources are virtual machines. When a job is submitted, a virtual workspace is created for the job by assigning existing virtual machines to process it. During the execution of the job the virtual machines are owned by the user and the user has access to his or her unique workspace image through the NFS-based distributed virtual file system [85–87]. An automatic virtual machine creation project called VMPlants [88] is provided with In-VIGO. VMPlants is used to automatically create custom root file systems to be used in In-VIGO workspaces.

4. ENABLING VIRTUAL ENVIRONMENTS IN A SHARED INFRASTRUCTURE

4.1 Introduction

With the advances in cyberinfrastructure technologies, considerable opportunities have been created for a broad spectrum of distributed and parallel computing applications to take advantage of massive amounts of aggregate computational power available through the Internet. Spanning multiple domains, a cyberinfrastructure aims to provide for the federation, allocation, and management of heterogeneous networked resources and makes them available to a large number of users. As such, large portions of the global computing community are joining together to form wide-area shared cyberinfrastructures, realizing the vision of global sharing and access.

Research challenges exist in fulfilling the full potential of such a shared cyberinfrastructure. The users of traditional resource sharing technologies, such as Beowulf Clusters [75] and the Grid [30, 79, 80, 89, 90], are familiar with the traditional *job submission and execution* model as well as the service-oriented access model as defined by the Open Grid Services Architecture (OGSA) [79]. Powered by technologies such as the Grid Resource Allocation and Management (GRAM) of Globus [91], the Grid provides a single detail-hiding interface for requesting and using networked resources for the execution of jobs submitted by various users. These job and service models have been widely used and they define an appropriate paradigm for resource sharing and program execution. Applications exist however, that exhibit an operational rather than functional nature, making it difficult to map these applications to independent jobs or services. An example is the fine-grain emulation of real-world systems such as airport operations and anti-terrorism exercises, each of which involves a large number of dynamic and diverse objects, contexts, and object interaction patterns.

Furthermore, applications may require specially configured and customized execution environments, including operating system and network level services as well as application-level services, packages, and libraries. For example, many scientific applications require mathematical and communication libraries such as Basic Linear Algebra Subroutines (BLAS) [92] and Message Passing Interface (MPI) [93], and many Java applications insist on a specific version of JVM. In a distributed infrastructure, such specific requirements may not always be mutually accommodated. Moreover, the requirements may be in conflict with each other (e.g., different versions of the same library).

Finally, it is not possible to prevent users from running applications that are untrustworthy or potentially mal-functioning. Software bugs and vulnerabilities may be introduced into an application either inadvertently or deliberately. For example, security vulnerabilities have been identified in well-known applications such as SETI@Home [78]. These vulnerabilities could be exploited to launch network attacks against any machine on the Internet. As a result, it is critical to contain any security impact incurred by an application, so that other applications, as well as the underlying shared infrastructure, will not be affected precipitously.

It is clear that the need exists for mutually isolated distributed environments on shared infrastructure as a complement to the job and service-oriented sharing model. To address the problems discussed above, mutually isolated distributed environments should have the following properties: (1) on-demand creation, (2) high customizability and configurability, (3) binary compatibility for applications, and (4) containment of negative impact of malicious or mal-functioning applications.

In its renaissance virtualization has become a promising technology for enabling virtual environments. Virtualization introduces a level of indirection between applications and the shared infrastructure. Technologies exist for the virtualization of machines and networks. Examples of virtual machine (VM) enabling systems include VMware [9], User-Mode Linux (UML) [7], and Xen [6]. Despite implementation differences, these systems all enable virtual machines that achieve binary compatibility

and networking capability just as do real machines. Examples of virtual network enabling systems include VNET [17], IPOP [60], and ViNE [61] all of which create virtual IP networks for confined virtual machine communications. However, VIOLIN [15] creates virtual environments by introducing additional wide-area isolation and adaptation capabilities.

The application of virtual machine technology to Grid computing was first proposed by Figueiredo, Dinda, and Fortes [11]. They have identified six major advantages of virtual machines for the Grid, namely security and isolation, customization, legacy support, administrator privileges, resource control, and site-independence. Their projects, In-VIGO [10] and Virtuoso [14], are among the first to address Grid resource virtualization and heterogeneity masking using virtual machine abstraction. The VMPlants architecture [88] within In-VIGO enables creation of customized virtual machines that exist in a shared execution environment adhering to the traditional job submission and execution model.

Taking virtualization even further, we have developed the VIOLIN middleware to enable isolated virtual environments on a shared cyberinfrastructure. The goal is to provide applications with customized and consistent runtime and networking environments with strong isolation from each other. The rest of this chapter presents the desirable features, experimental results, and ongoing work towards this goal.

Figure 4.1 gives a multi-layer overview of virtual environments on a shared physical infrastructure enabled by VIOLIN. The bottom layer is the physical infrastructure with heterogeneous networked resources spanning multiple network domains. The middle layer is the enhanced infrastructure integrated and managed by value-added middleware systems. Our middleware for virtual environments is deployed in this layer. The top layer consists of mutually isolated virtual environments, each with its own network, operating system, and application services customized for the applications running in it. They are employed by users or user groups with various computational needs.

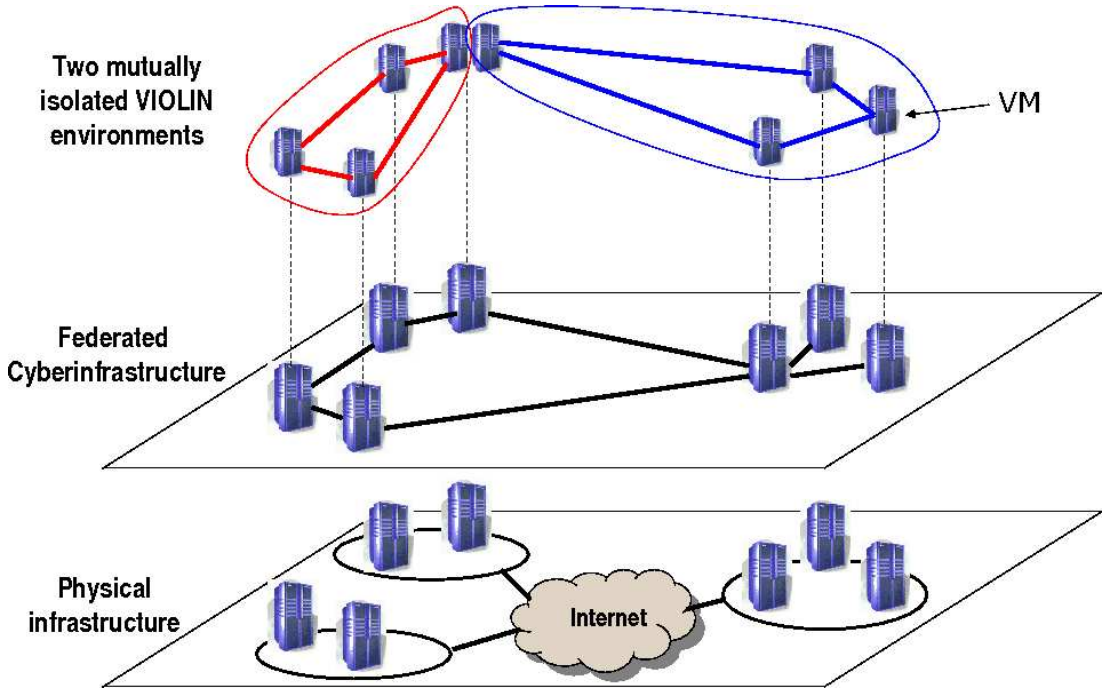


Fig. 4.1. Multi-layer overview of virtual distributed environments on a shared cyberinfrastructure

Virtual distributed environments are supported by the integration of virtual network and on-demand virtual machine creation and customization technologies. This leads to the VIOLIN middleware system we developed that enables virtual distributed environments.

The four desirable properties of virtual distributed environments mentioned in Chapter 1 will be realized as follows:

- On-demand creation of VIOLIN environments involving both virtual machines and the virtual IP network connecting the virtual machines.
- Customization of VIOLIN environments including the virtual network topology and services, operating system services, and application services/packages/libraries.
- Binary compatibility is achieved by creating the same runtime and network environment under which an application was originally developed.

- Containment of negative impact is achieved by isolating virtual network address space, limiting both virtual machine resources and inter-virtual machine traffic rate, and granting users administrative privileges within the virtual environments without requiring privileges at the physical infrastructure level.

4.2 Virtual Machines and Virtual Networks

We first survey techniques related to networking support for virtual machines. Current virtual machine technologies can be used to create localized virtual private environments. Virtual machines can be requested and instantiated on-demand, each with a regular IP address as well as a customized installation of operating system and application services. Enhancing the underlying host's operating system allows machines to support multiple virtual machines each of which are guaranteed a 'slice' of the physical host with respect to CPU, memory, and network bandwidth. Typically, the modified operating system that supports multiple independent virtual machines is called a Virtual Machine Monitor.

Figure 4.2 shows a typical host that is running a VMM which is supporting virtual machines. Virtual machines reside on a physical machine or *host*. A virtual machine monitor sits between the host's hardware and the virtual machines. The VMM directly interfaces with the hardware and provides each VM with its own virtual hardware. The virtual machine's main responsibility is to manage the multiplexing of the real hardware into multiple sets of virtual hardware for the virtual machines. The operating systems of each virtual machine logically interface with their own virtual hardware and are unaware that they are sharing physical hardware with other virtual machines. As a result, multiple virtual machines support multiple users, processes, and applications, and can utilize a single piece of hardware while maintaining completely isolated and independent operating systems, configurations, and user space.

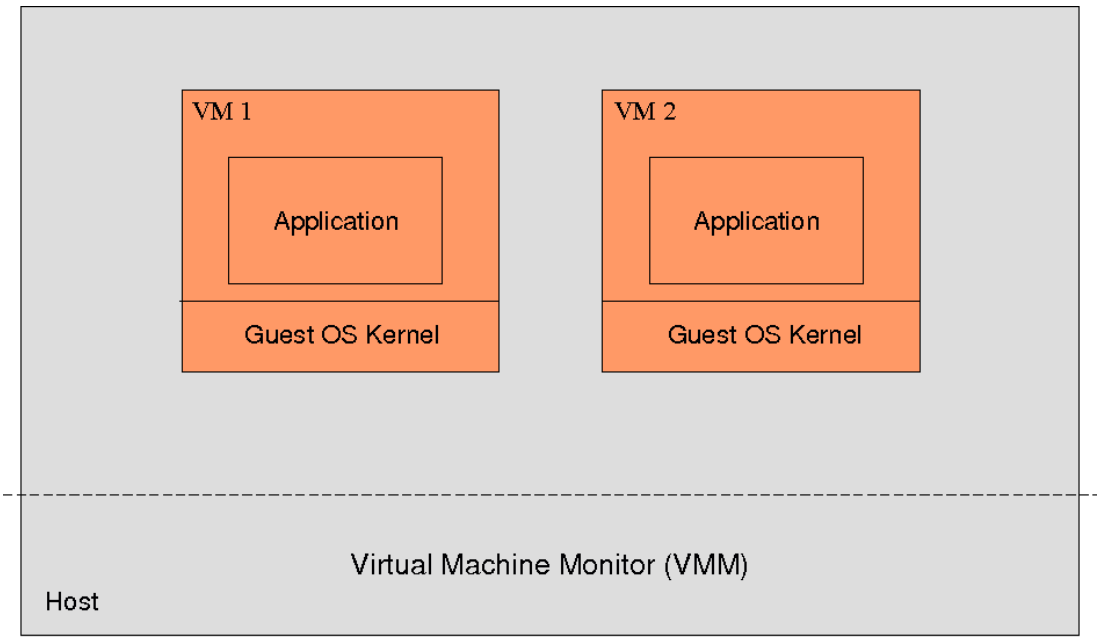


Fig. 4.2. Typical virtual machine configuration including the virtual machine monitor and two virtual machines.

Using the technology described above, however, isolation is achieved only between individual virtual machines. A set of virtual machines does not automatically create an isolated virtual environment. Typically, virtual machines use shared networks which cause them to be addressable to/from other Internet hosts. Moreover, it is difficult to manage multiple virtual networks because all the virtual machines share the same IP address space. As a result, conflict may occur between these virtual networks and administration of multiple conflicting virtual networks must be performed by a single administrator.

Although virtualization makes use of multiplexing at the operating system level, networking is established without considering isolation. Figures 4.3-4.5 show how virtual machines are commonly networked. There are three types of networking techniques that are possible using currently available off-the-shelf virtual machines.

- **Host-only networking**. Figure 4.3 shows host-only networking. As in all types of virtual machine networking, the virtual machine has a virtual NIC

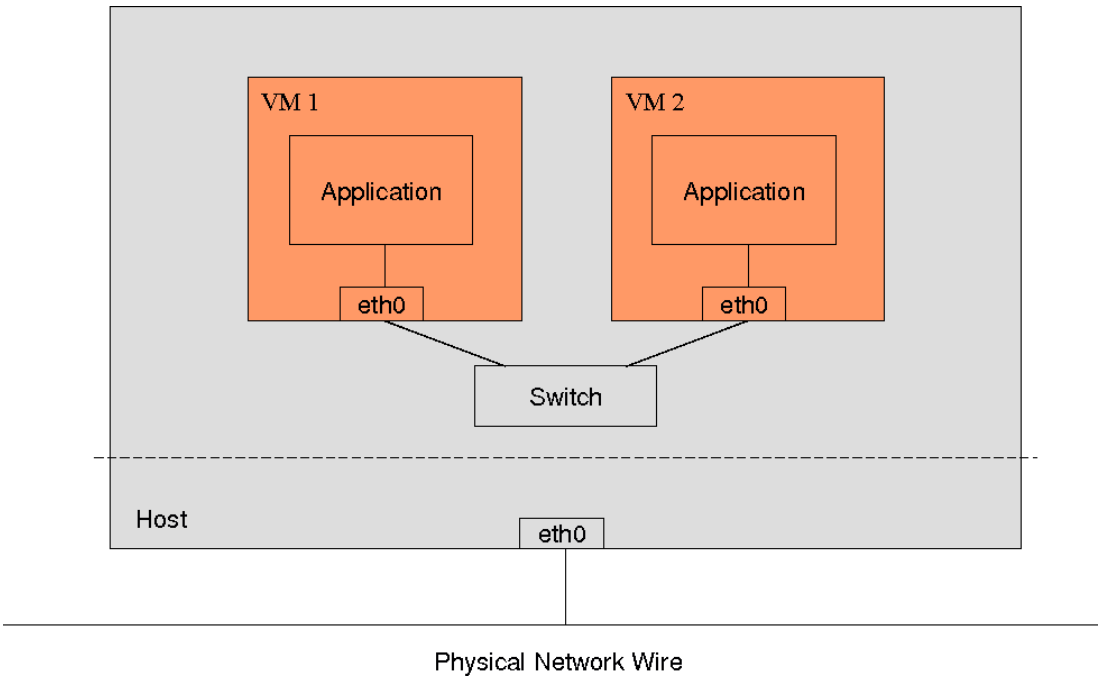


Fig. 4.3. Virtual machines using host-only networking.

provided by the VMM. In host-only networking the virtual NIC on each virtual machine is linked to a virtual switch residing on the host. The virtual switch is software that sits below the virtual machine and above the VMM. It is important to note that each virtual machine has its own operating system including it's own network stack. The virtual switch communicates with virtual NICs by sending and receiving Ethernet frames. In this configuration, the virtual machines communicate through a virtual network that is isolated from the underlying physical network. The limitation of host-only networking is that only virtual machines on a single host can communicate through the switch. This is analogous an isolated LAN that is not connected to any WAN and has wires that are only long enough to reach machines in a single room.

- **Network Address Translation (NAT) networking**. Figure 4.4 shows NAT networking. In this configuration the host becomes part of the host-only network and acts as a NAT router forwarding IP packets from the virtual machines to

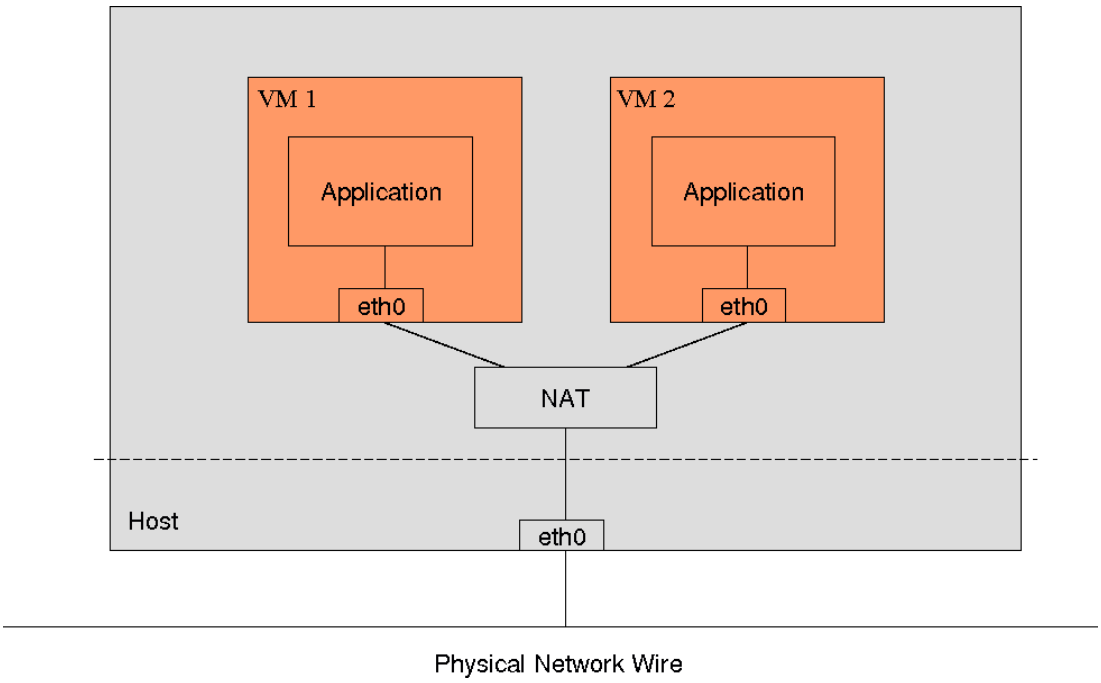


Fig. 4.4. Virtual machines using NAT networking.

the physical network. As in all NAT routing, the virtual machines can connect to services outside of the host, but they cannot host services and make them accessible by external machines. This is analogous to a small LAN located in a private residence. The computers on the LAN are confined to the house and can access Internet services through a NAT router connected to a DSL or Cable ISP.

- **Pass-through networking**. Figure 4.5 shows pass-through networking. In pass-through networking the host uses a *bridge*[1] to extend the physical network to include the host-only network. The host is responsible for listening to the physical network for all Ethernet frames and for forwarding them to the internal virtual machines. Using pass-through networking, the virtual machines can both access the outside network as well as host services that can be accessed from outside of the host. The drawback of using pass-through networking is

[1]A bridge is a standard Linux facility typically used to daisy-chain multiple physical Ethernets

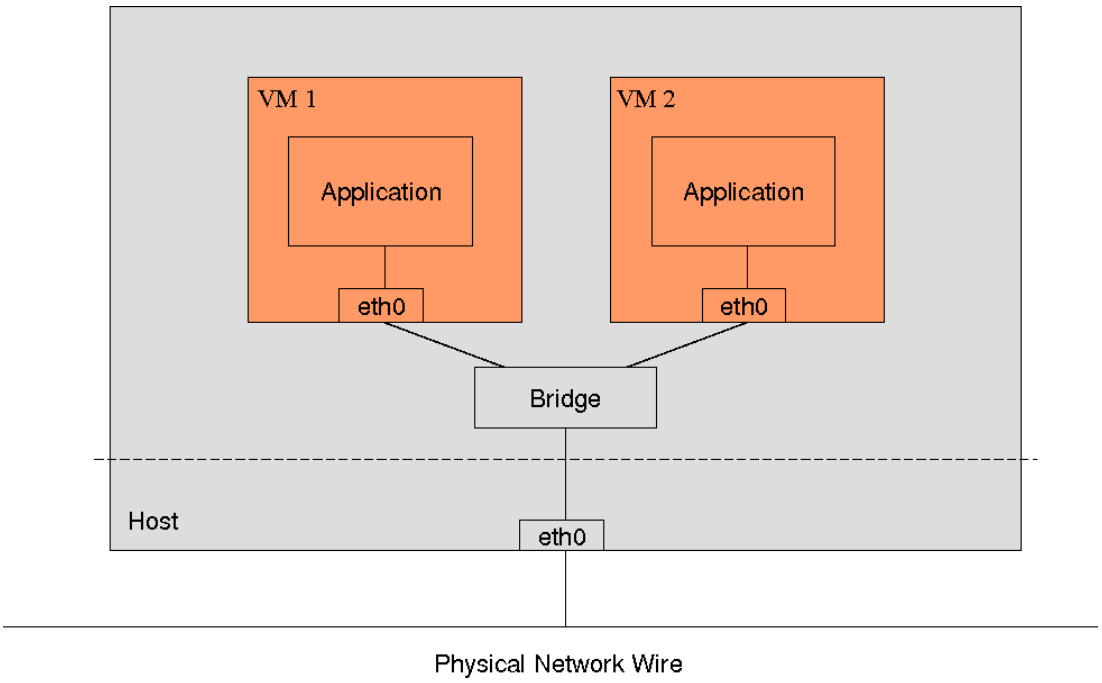


Fig. 4.5. Virtual machines using bridged networking.

that each virtual machine needs to have an IP address from the host network's IP address space. In situations where large numbers of virtual machines are needed the address space may become exhausted. This scenario is analogous to daisy-chaining the host only network to an existing physical network.

All three of these virtual machine networking techniques are useful for many applications of virtual machines; however, they all have limitations that prevent their use for creating isolated virtual environments that span the cooperating domains of a shared cyberinfrastructure.

4.3 Design of VIOLIN Networking

The vision of VIOLIN is to enable adaptive isolated virtual environments distributed across a cyberinfrastructure. The key properties that VIOLIN must provide are: (1) to federate groups of cyberinfrastructure resources to be utilized for a single

purpose, (2) to maintain isolation between virtual environments and the host infrastructure, and (3) to enable the adaptive properties of virtualization in a infrastructure distributed across domains. None of these properties is possible with the standard virtual machine networking techniques described above.

To meet these challenges, VIOLIN uses a novel fourth category of virtual machine networking we call *tunneled networking* or *VIOLIN networking*. In contrast to standard virtual machine networking, VIOLIN networking maintains the isolation properties of host-only networking while allowing virtual machines in the VIOLIN network to be distributed across the Internet.

We introduce a level of communication indirection between the virtual machines and the underlying infrastructure that decouples the virtual environment from the underlying infrastructure. Inside a virtual environment, the virtual machines communicate with each other using standard IP services. Below the virtual environment, Ethernet services are emulated via application-level tunneling mechanisms.

The effects of this design responds to the properties needed to support virtual environments on a cyberinfrastructure. VIOLIN networking enables groups of distributed virtual machines to be logically on a single LAN and execute arbitrary codes that require LAN connectivity. Encapsulation and tunneling of frames isolates these virtual environments. Adaptive virtualization mechanisms that require LAN connectivity, such as live migration, can be performed over a VIOLIN network even though the network may span multiple domains.

In addition, such a design leads to an important benefit: the VIOLIN environments can utilize network services that are not widely deployed in the real Internet (e.g., IP multicast and IPv6) and enable these network services in the virtual network. By virtualizing the network below layer-2, VIOLIN can support any high-level network protocol including non-standard ones not supported by the underlying infrastructure.

VIOLIN can be used to create isolated virtual environments on a shared cyberinfrastructure because it creates high-order virtual IP networks for virtual machines. These virtual networks have the following salient features:

- Each virtual network has its own IP address space. This means that the address spaces of different virtual networks can safely overlap allowing for independent administration of multiple virtual environments sharing the same host infrastructure.

- Virtual machines in a virtual network are not visible on the Internet, preventing attacks from the virtual environments to the Internet as well as direct attacks from the Internet to the virtual environments.

- The traffic volume between specific pairs of virtual machines is bounded by a user-specified threshold, preventing the generation of excessive traffic from ill-behaving virtual machines. Furthermore, the enforcement of both virtual network topology and traffic volume is untamperable from inside the virtual machines.

- The layer-2 virtualization of VIOLIN provides virtual Ethernet to be stretched across domain boundaries. This feature allows for wide-area migration of virtual machine where the virtual machine logically remains within the original Ethernet. This ability places VIOLIN as one of the few network virtualization mechanisms that can migrate live virtual machines without modifying network configuration and disrupting the applications running inside.

The design of VIOLIN employs a novel technique of networking that is analogous to a distributed version of host-only networking. Figure 4.6 shows a VIOLIN network. As in host-only networking, VIOLIN virtual machines connect to an isolated virtual layer-2 network; however in VIOLIN, virtual machines from multiple hosts can connect to the same layer-2 switch. As the figure shows, a VIOLIN virtual network can be viewed as independent LANs composed of virtual machines from multiple hosts.

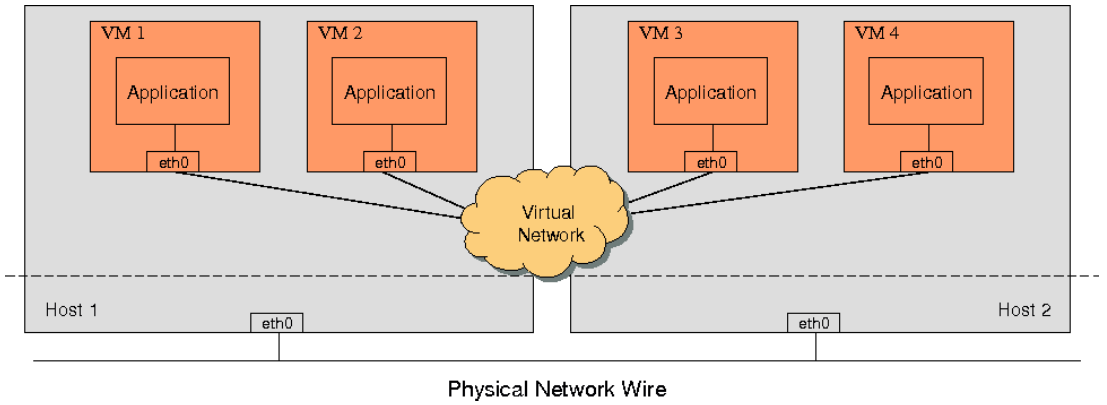


Fig. 4.6. Virtual machines connected to a virtual VIOLIN network.

4.4 Implementation of VIOLIN

We have implemented a middleware system that enables VIOLIN virtual distributed environments. Originally, the core features of VIOLIN were implemented exclusively for User-Mode Linux virtual machines [15]. Over time, however, the performance provided by machine level paravirtualization led to the incorporation of a *tun/tap* interface that supports connectivity with many virtualization platforms. With the addition of the tun/tap interface, VIOLIN can support Xen and VMware which are the most popular virtualization platforms used in academia and industry.

4.4.1 VIOLIN Core

The core feature of VIOLIN networking is the ability to connect virtual machines residing on distributed hosts with a virtual LAN. Network traffic on the virtual LAN is isolated from the underlying infrastructure and virtual machines do not have a presence in the underlying physical network. In contrast to traditional virtual machine networking techniques, virtual machines in a VIOLIN network connect to a distributed virtual network switch. The virtual switch is composed of multiple daemons one residing on each host participating in the VIOLIN network. Together the

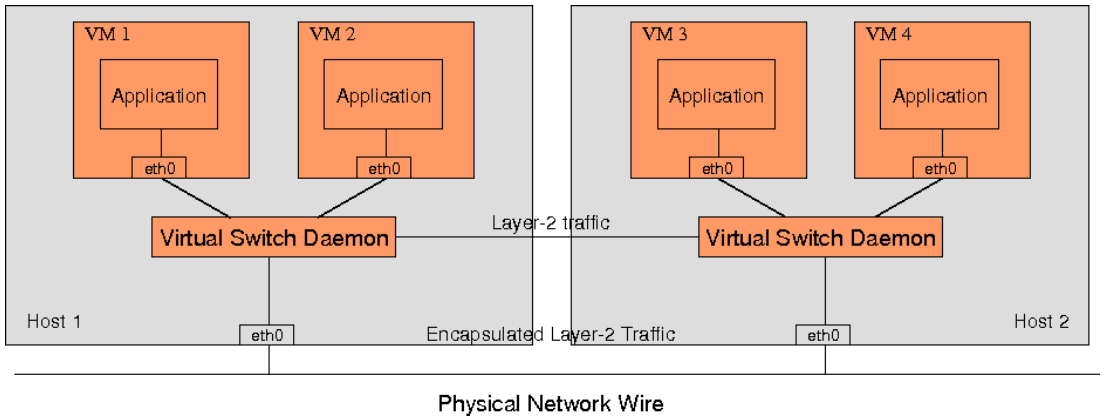


Fig. 4.7. Virtual machines connect to Virtual Switch Daemons which tunnel traffic between hosts.

daemons form a peer-to-peer underlay network that emulates a single layer-two switch of the VIOLIN network.

Figure 4.7 depicts the core functionality of VIOLIN networking. In the figure there are two hosts and four virtual machines. The virtual machines connect to a VIOLIN switch daemon on its host. The switch daemons together compose a single distributed virtual switch which forwards virtual layer-2 network traffic (namely the Ethernet frames). This demonstrates how VIOLIN encapsulates virtual network traffic and maintains isolation between the VIOLIN network and the underlying physical network.

From the point-of-view of the virtual machines, the VIOLIN network is an Ethernet LAN that is accessible through the local VIOLIN switch daemon. The virtual machines send and receive Ethernet frames to and from the virtual switch as if it were a physical machine communicating with a physical switch. The virtual machines do not know where their communication partners are physically located.

There are two planes of communication used by the VIOLIN switch. The first is the *data plane* that is used to transport the Ethernet frames between the virtual machines and the second is the *control plane* that is uses to transmit control and signaling messages among the distributed switch daemons.

Data Plane

The VIOLIN data plane is a tunneling peer-to-peer underlay. It is solely responsible for obtaining and transporting Ethernet frames between virtual machines. In terms of the ISO 7-layer reference model, VIOLIN comprises the *Physical Hardware Connection* layer (layer-1) and *Data Link* layer (layer-2). Figure 4.8 shows the network stack as seen by the virtual machines. VIOLIN is implemented at the Link layer and lower with respect to the virtual machines in order to allow the deployment of any higher-level network protocols (e.g. IP Multicast, IPv6, or experimental protocols) to be used within VIOLIN environments.

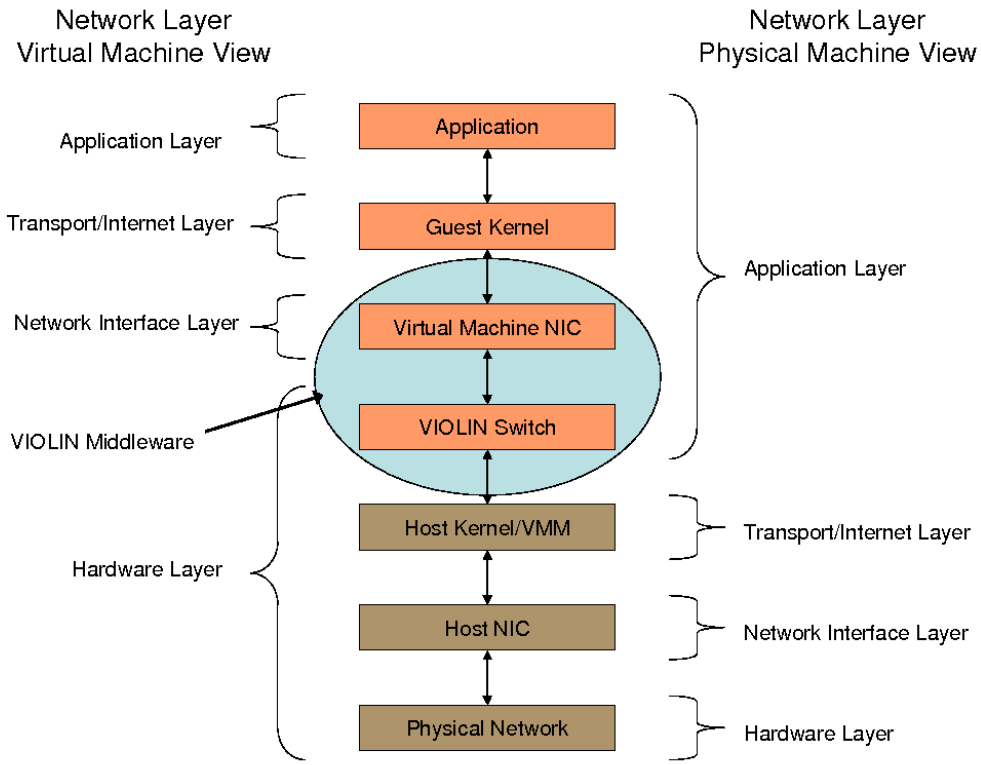


Fig. 4.8. Network layers from the view point of both the virtual and physical machines.

In contrast, from the point of view of the host infrastructure, the VIOLIN data plane is at the *Application* layer (layer-7). The choice of application layer implementation allows for complete network isolation between the VIOLIN environments and the underlying host network. Network frames are encapsulated within UDP packets and tunneled between the virtual switch daemons. The physical network never directly handles frames from the virtual network. In addition, each VIOLIN network exists in an isolated IP space giving users flexibility to apply any network setting they want without fear of conflict with other virtual networks or the underlying physical network. For example, multiple VIOLIN environments sharing the same infrastructure can use IP addresses (or even MAC addresses) from the same address space.

Control Plane

The second communication plane used by VIOLIN daemons is the *control plane*. The set of VIOLIN switch daemons emulates a single layer-2 switch. The control plane is used to organize the daemons and update virtual machine location information.

A physical Ethernet switch has many *ports* through which machines connect to the network. The machines send Ethernet frames to the switch through the ports. The switch obtains the frame's destination using the MAC address in the Ethernet header. Based on the destination address, the switch forwards the frame to the outgoing port found in its look-up table.

VIOLIN switches mimic the functionality of physical switches, however, their distributed nature and the dynamics of virtualization create additional challenges. First, the distributed switch must be able to adapt to infrastructure changes (e.g., node failures and the addition of new daemons and virtual machines). Second, each daemon composing the virtual switch must be able to adapt to changes in the virtual environment (e.g. the addition, deletion, or migration of virtual machines).

To accomplish these tasks the switch daemons are organized into a peer-to-peer network. Each switch daemon maintains a control channel to every other switch

daemon. The control communication channel is independent of the data plane connections. The control plane connections are TCP connections between each pair of switch daemons. It is important to note that any peer-to-peer scheme will work and VIOLIN has all of the benefits and liabilities of modern peer-to-peer techniques. For performance reasons, the only important attribute of VIOLIN routing is that the data plane maintain direct connections between any pairs of daemons that have virtual machines that are communicating. In other words, data plane traffic must only take one hop through the peer-to-peer network.

4.4.2 User-mode Linux Interface

Although VIOLIN's features can be implemented in most virtualization architectures, it was originally implemented for User-Mode Linux virtual machines. The intent was to take advantage of UML's already existing user level execution and open-source code to aid our implementation.

Standard UML virtual machines are instantiated as processes in a Linux user's execution space. Communication outside of the host machine is possible through a virtual network interface, called a tap device, residing in the host's kernel. The UML virtual machine contains a virtual network interface that connects to the host's tap device, and the host acts as a router forwarding packets between the virtual machines and onto the physical network. Root level privileges on the host are needed to safely create tap devices and manage the host's routing tables that enable their use.

VIOLIN bypasses the need for tap devices when using User-Mode Linux and allows virtual machines to exist in a private IP space logically disconnected from the physical network. Figure 4.9 depicts a VIOLIN enabled UML virtual machine. The virtual machine contains a virtual network interface that does not connect to the host, but instead maintains a socket connection between the virtual machine's host process and a virtual switch daemon existing on any host connected to the physical network. Placing a virtual switch daemon on each host increases performance but

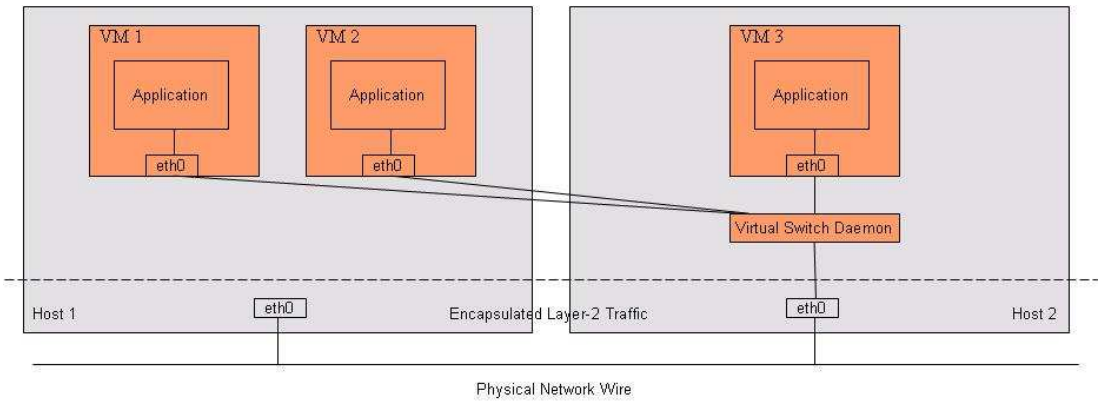


Fig. 4.9. UML virtual machines connect directly to virtual switch daemons located on any accessible host.

is not strictly necessary for UML virtual machines. The virtual switch behaves like a physical switch accepting connections from many virtual machines and forwarding network packets to their destination virtual machines.

As opposed to the more general interface described in section 4.4.3, the UML interface allows for virtual environments that run completely in user space with respect to the hosts. User space virtual environments have the benefit of running without host-level administrative privileges.

4.4.3 Generic Interface (tun/tap)

Since the initial UML implementation of VIOLIN, more powerful, better performing virtualization platforms have matured and become more popular in academic communities. In response, we have implemented a more universal interface to VIOLIN. This interface relies on a kernel-level tun/tap device and supports most common virtualization platforms, most importantly Xen and VMware. Although tun/tap devices create a more generic VIOLIN interface they have the drawback of requiring some configuration by the administrator and an increased network overhead. It is possible to modify each virtualization platform to bypass the tun/tap devices and connect directly to the switch daemon similar to the UML interface; however, perfor-

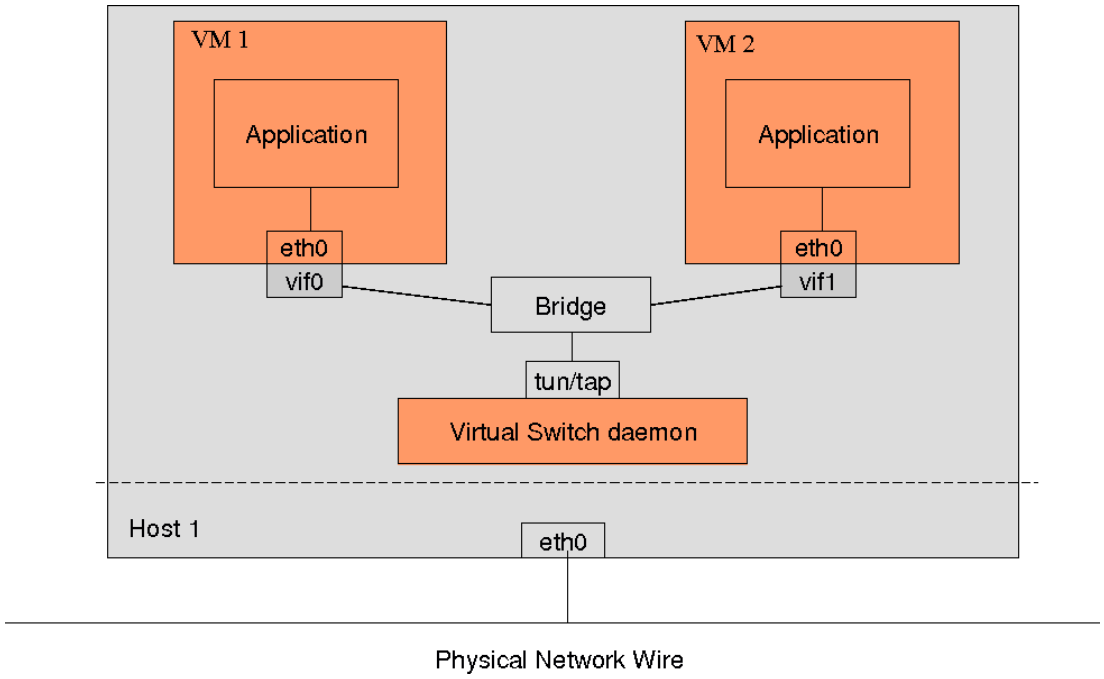


Fig. 4.10. The tun/tap interface to VIOLIN supports connections form many popular virtualization platforms.

mance evaluation shows that even with the tun/tap device, the overhead of VIOLIN is low, and a wide-array of virtual machines can use the interface making it an attractive alternative to modifying each virtual machine platform to interface directly with VIOLIN.

A common approach to networking virtual machines is to have a two-part device driver. The inner half of the device driver is a virtual NIC seen by the guest operating system while the outer half appears as a virtual NIC in the host operating system. The two halves of the device shuttle frames in and out of the virtual machine. Typically, the host uses standard bridging or routing mechanisms to connect the outer virtual device to a network. As seen in Figure 4.10, the tun/tap interface to VIOLIN uses a similar approach by bridging the virtual machine's device driver to a tun/tap device. A tun/tap device is an operating system device driver that has the outward appearance of the NIC, but forwards frames to a user-level process instead of a physical network. In VIOLIN, the tun/tap device forwards frames to the the virtual switch

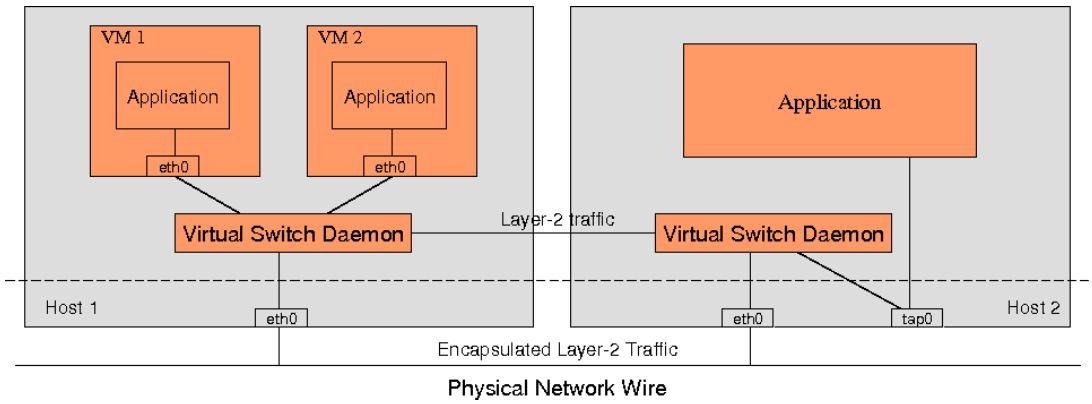


Fig. 4.11. The tun/tap device interface allows for *hybrid* VIOLIN environments composed of both virtual and physical machines.

daemons. The switch daemons handle the frames using the same methods as they handle frames originating from a UML interface.

As an additional advantage, the generic mechanisms used by the tun/tap interface to VIOLIN are flexible enough to allow physical machines to connect to a VIOLIN network. As shown in Figure 4.11. Virtual and physical machines can connect to the network through the virtual switch daemons. To a physical machine a tun/tap device functions as any network device driver. In order for a physical machine to connect to a VIOLIN network, the machine must modify its IP routing table to route the desired traffic to a tun/tap device associated with a VIOLIN switch daemon. Traffic from the physical machine will be routed to the switch through the tun/tap device and the switch will handle the traffic as it handles all other traffic.

Connecting a physical machine to a VIOLIN network requires some additional administration. Although the machine will be connected to the VIOLIN network it will also be responsible for the execution of the switch daemon and the tunneling of frames. For this reason the physical machine must retain its physical network interface and the associated routing tables. As a result the physical machine will be a fully connected to both networks. The effect is that the IP address space of the VIOLIN network and the physical network must be disjoint. On the other hand, in

purely virtual VIOLIN networks the IP address spaces are independent and can safely overlap.

4.4.4 Multiple VIOLIN Environments Sharing Resources

The fundamental benefit of applying VIOLIN to shared infrastructures is network-level isolation. Figure 4.12 demonstrates VIOLIN's ability to achieve mutual isolation between virtual environments sharing the same infrastructure and even the same hosts.

In the figure, there are two isolated virtual environments: the blue environment and the orange environment. In order to maintain isolation, each environment must have its own distributed network switch. In turn, each host must have a switch daemon for each of the environments. Notice that there are independent bridges associated with each switch daemon. The blue virtual machines connect to the blue switch daemons through the blue bridges while the orange virtual machines connect to the orange switch daemons through the orange bridges. The blue and orange switch daemons form two independent peer-to-peer underlay networks, one for switches of each color. Although both virtual environments forward tunneled traffic over the same underlying infrastructure they remain logically isolated LANs. Thanks to their mutual isolation, all concurrently running VIOLINs use the same set of IP addresses for their virtual machines without causing any conflict and are mutually isolated with respect to traffic and network services.

4.5 Performance Measurements

VIOLIN is intended to provided many non-performance related value-added features to virtual environments running on a shared cyberinfrastructure. However, performance is crucial to the success of VIOLIN in supporting real-world scientific applications. We present two experiments measuring VIOLIN performance that show that the performance overhead of using VIOLIN for high-performance parallel and

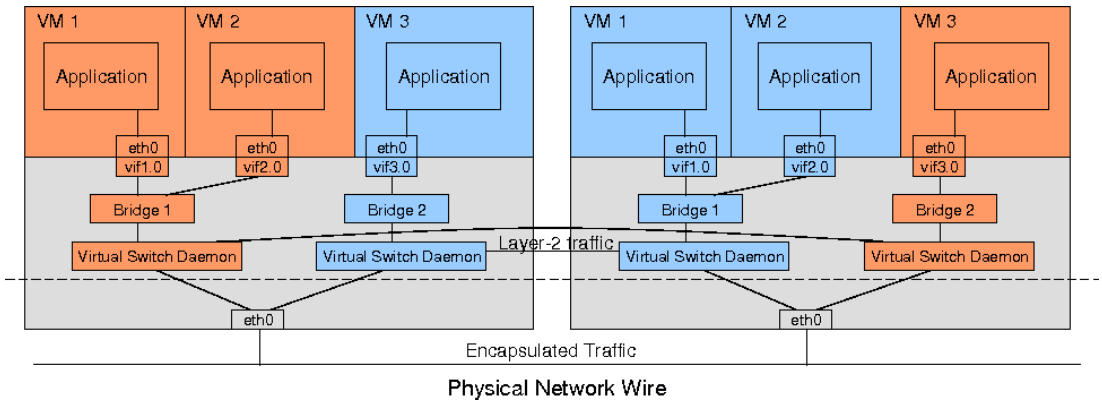


Fig. 4.12. Multiple VIOLIN environments maintain isolation even when sharing the same physical resources.

distributed applications is low. The first experiment uses the original UML virtual machines, while the second uses better performing Xen virtual machines.

4.5.1 High-performance Linpack in a UML-based VIOLIN Environments

In this section, we present experimental results from virtual environments restricted to hosts from a single domain. The goal is to demonstrate performance of UML-based VIOLIN environments compared with that of the underlying physical cluster. The cluster consists of hosts each with dual 1.2GHz Athlon processors and 1GB RAM running Debian Linux 3.0. The hosts are connected by a 100 Mb/s Ethernet switch. The application running in each VIOLIN environment is the High Performance Linpack (HPL) [94] benchmark. Our purpose is to stress a UML VIOLIN environment to its limit by finding the maximum number of floating point operations per second (Flops) it can achieve relative to the Flops achievable without using virtualization.

The VIOLIN virtual cluster cannot out-perform the underlying physical cluster, however, a small loss of performance will justify using virtualization. We aim to show that virtualization has a small performance penalty. In this experiment, we run HPL on an increasing number of processors in a physical cluster and compare the FLops

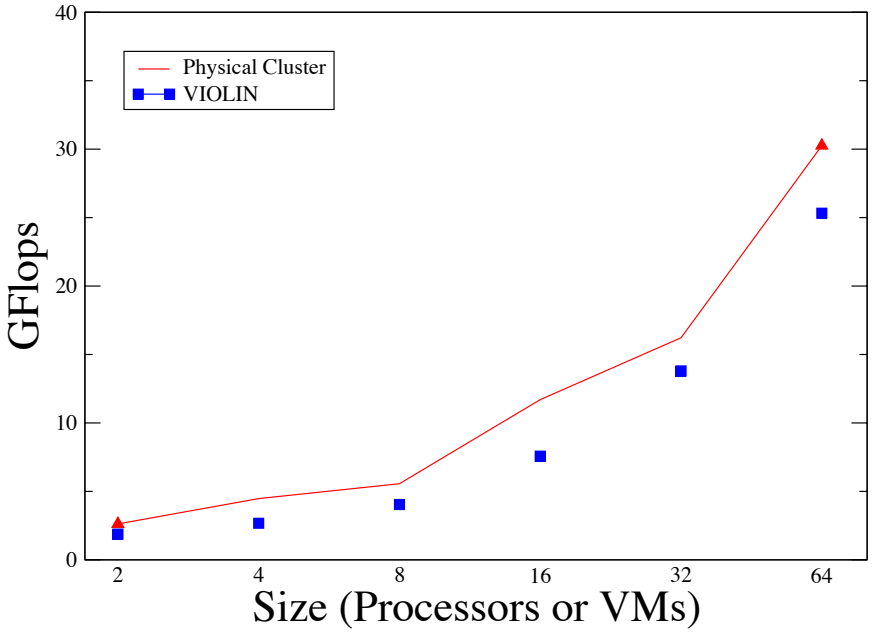


Fig. 4.13. Comparison of UML VIOLIN with physical cluster

achieved with that of the same problem specification on the same number of virtual machines in a VIOLIN environment. For each virtual and physical cluster size we find the maximum possible performance by tuning HPL parameters, most notably: problem size, block size, and process grid shape. There is a non-zero amount of memory overhead using virtualization. This overhead reduces the maximum problem size that can be run in a virtual environment. Increasing the problem size to use all available memory would not result in a fair comparison. Therefore, the problem size parameter is tuned to the maximum problem size that can be run by the VIOLIN environment and is used for both the virtual and non-virtual test of the same virtual and physical cluster size. All other HPL parameters are tuned for maximum performance for each environment. Also, because the host machines have dual-processors, two MPI processes and two virtual machines per host are instantiated for the physical cluster and VIOLIN experiments, respectively.

Figure 4.13 shows the experimental results of running HPL on up to 64 processors or virtual machines. From this figure we see that the performance achievable when

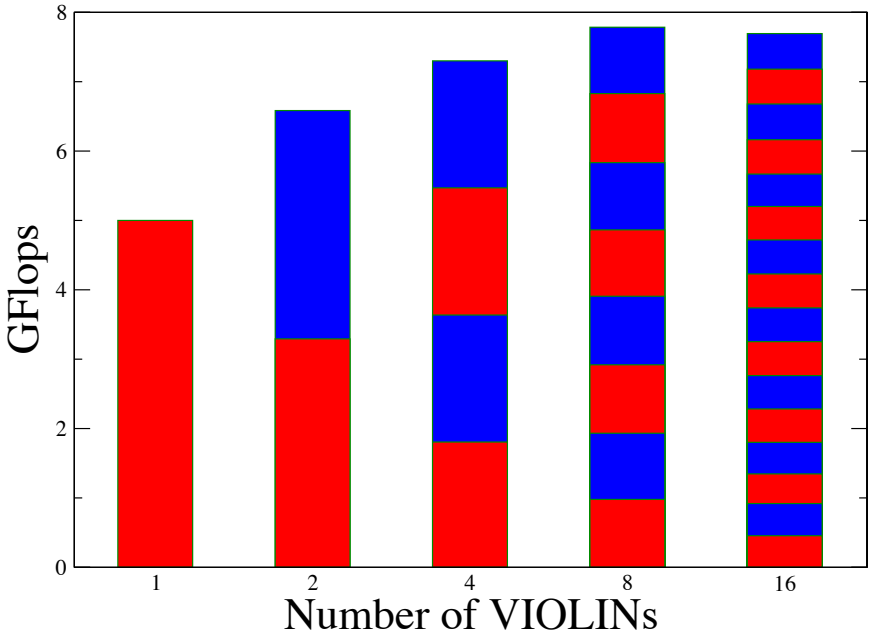


Fig. 4.14. Measured performance of multiple UML VIOLINs concurrently running on shared hosts.

running HPL in a virtualized VIOLIN environment is about 85% of that running on the physical machine without virtualization. In addition this trend scales to a cluster size of at least 64 nodes. This performance penalty is small enough to justify the value-added features of virtualization for many applications and the scaling trend shows promise for the use of virtualization in larger environments.

The results in figure 4.14 show that as more virtual environments share available resources each virtual environment receives approximately an equal share of the resources and the aggregate performance of all virtual environments is affected very little. We observed an increase in aggregate performance up to 8 virtual environments sharing the same set of hosts followed by a very small decrease at 16 virtual environments. These results show that there is little overhead (namely, loss of aggregate performance) while increasing the number of UML virtual environments up to 16.

4.5.2 NEMO3D in Xen-based VIOLIN Environment

In this section we present another set of experimental results from virtual environments confined to a single cluster. The goal, again, is to demonstrate performance of VIOLINs compared with that of the underlying physical cluster. The difference, however, is that this experiment uses the more powerful and efficient Xen 3.0 virtual machines and the tun/tap interface to VIOLIN. The Xen experiment uses a cluster consisting of Dell 1425s with 2GB of RAM and two hyper-threaded Xeon processors running at 3.00 GHz[2]. The application running in each VIOLIN environment is NEMO3D [95]. The purpose is to find the overhead of using Xen-based VIOLIN environments on a real MPI application.

Several virtual environment configurations were used:

- **Configuration 1**: One virtual machine per host.
- **Configuration 2**: Two virtual machines per host with 'locality'. Locality is defined as placing virtual machines on hosts such that the pairs of virtual machines sharing a host are 'neighbors' within the NEMO3D execution (i.e. virtual machines that communicate heavily). Neighbors in NEMO3D communicate significantly and placing them on the same hosts takes advantage of the fast communication within a host. We expect good performance using this allocation because much of the communication will be between virtual machines share a host which will limit the traffic tunneled through the physical network.
- **Configuration 3**: Two virtual machines per host without 'locality'. Virtual machines sharing a host are NOT 'neighbors' within the NEMO3D execution. They take advantage of both processors and always use the slower inter-host communication. This is the least optimal allocation of two virtual machines to each host, and increases the load on the VIOLIN switch daemons. We expect

[2]The cluster used in the previous experiment no longer exists so the Xen experiments are not directly comparably to the UML HPL experiments.

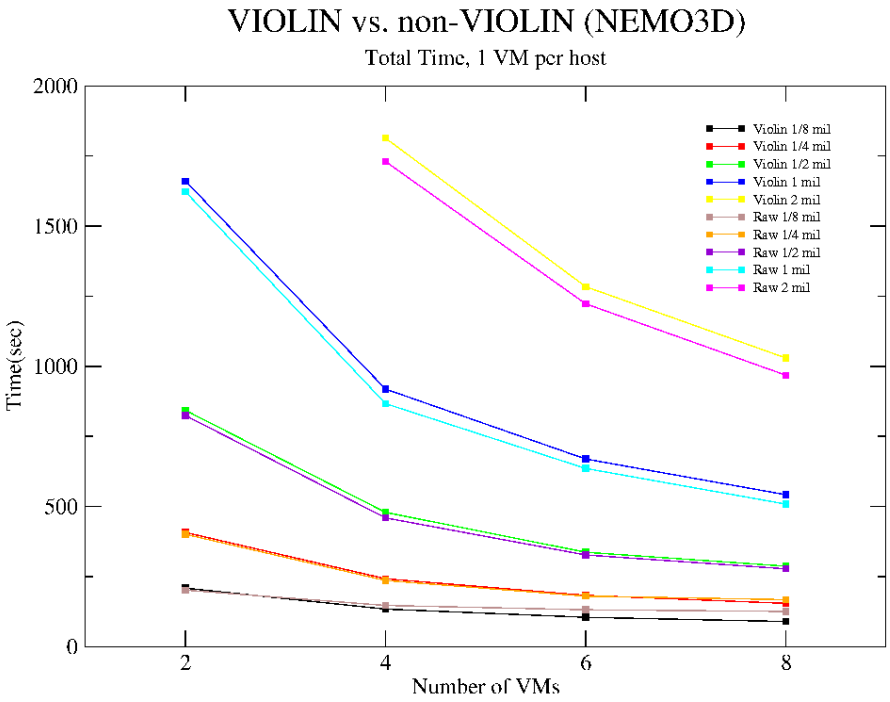


Fig. 4.15. Configuration 1: Performance of VIOLIN vs. non-VIOLIN environments running NEMO3D. One virtual machine or process per host.

poor performance using this allocation because all tunneled traffic will be sent through the physical network.

The total overhead seen in Figure 4.15 (configuration 1) shows that VIOLIN has an overhead of less than 6%. Compared to the UML VIOLIN environments in section 4.5.1 we see a significant reduction in overhead with the Xen-based VIOLIN environments. Many HPC applications can tolerate a 6% overhead considering the value-added benefits of VIOLIN.

Figures 4.16 (configuration 2) and 4.17 (configuration 3) show that as we increase the number of virtual machines sharing each host to two, the overhead increases. This is expected as the software VIOLIN switch daemons incur an increased processing load in order to handle twice as many virtual network frames. In addition, we know that

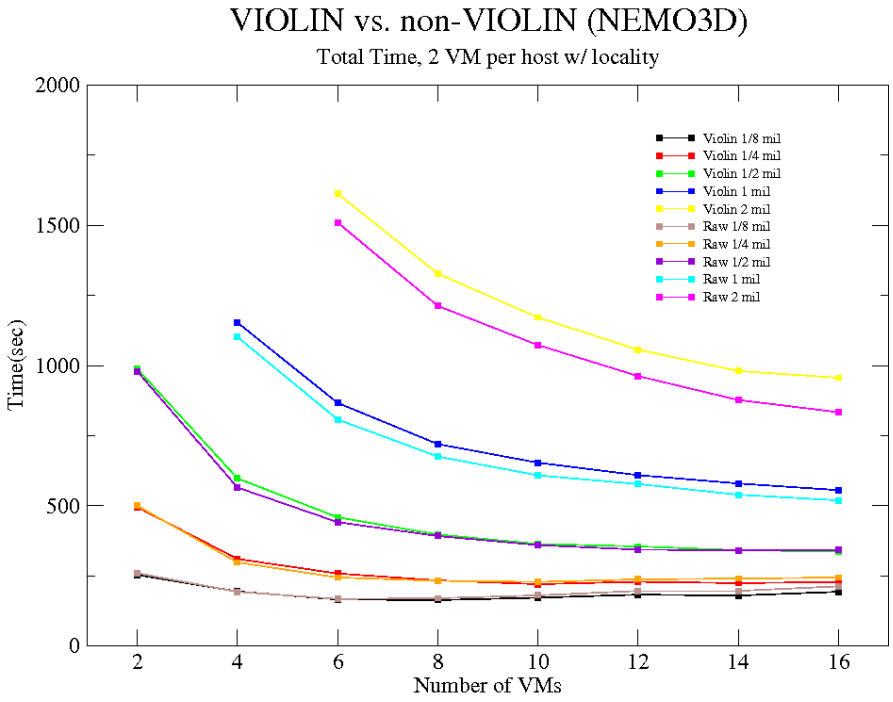


Fig. 4.16. Configuration 2: Performance of VIOLIN vs. non-VIOLIN environments running NEMO3D. Two virtual machines or processes per host with locality.

NEMO3D communicates in a ring pattern. Figure 4.16 shows the overhead of VIOLIN environments when the pairs of virtual machines assigned to each host are neighbors in the communication ring. Figure 4.17 shows the overhead of VIOLIN when the distribution of virtual machines is chosen such that neighbors in the communication ring are never paired on the same host. In both cases there is increase in overhead as network traffic is increased. However, comparing Figures 4.16 and 4.17, the configuration without correct locality has more overhead than the one with locality. It should be noted here that network and I/O overhead are the largest contributors to the overhead of Xen and that the Xen developers are focused on increasing I/O performance. We would expect this overhead to be reduced with future versions of Xen.

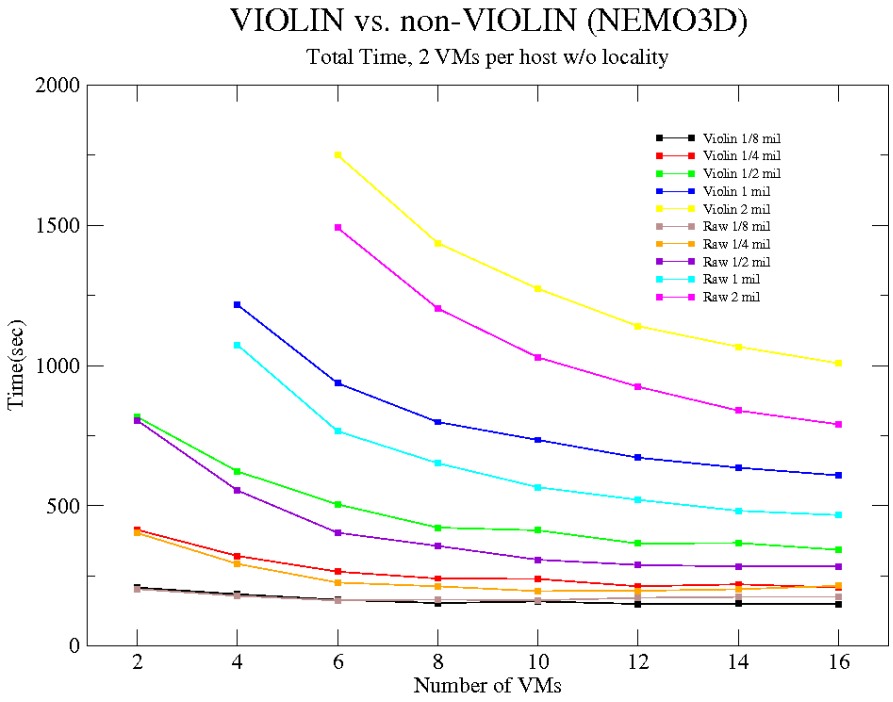


Fig. 4.17. Configuration 3: Performance of VIOLIN vs. non-VIOLIN environments running NEMO3D. Two virtual machines or processes per host without locality.

4.6 Summary

In this chapter, we advocate the creation of virtual distributed environments in shared cyberinfrastructures. The goal is to provide applications with their own customized runtime and networking environments with strong isolation from each other while sharing the resources in the same infrastructure. We advocate the integration and extension of virtual machine and virtual network technologies, in order to enable virtual distributed environments. Our VIOLIN-based middleware system demonstrates the feasibility and promise of deploying virtual environments for computational applications.. Experimental results urge further development and investigation of virtual distributed environments as a useful paradigm for sharing cyberinfrastructures.

5. VIRTUAL ENVIRONMENTS FOR INTER-DOMAIN CLUSTER SHARING

5.1 Introduction

As demonstrated in Chapter 4, VIOLIN virtual environments are isolated, secure, and customizable environments that use CPU, memory, and network recourses provided by a cyberinfrastructure. Chapter 4 briefly mentions dynamic properties of VIOLIN virtual environments, such as on-demand virtual machine creation, on-demand virtual machines destruction, resource scaling, and live migration. The remainder of this dissertation addresses using the dynamic properties of VIOLIN environments to adapt to the resource demands of the applications they are running. This chapter presents techniques that can be used to share traditional computer clusters within a campus. This chapter extends the concepts presented in the previous chapter by enabling the VIOLIN middleware to utilize the adaptive properties of virtualization. The challenge addressed is to use the adaptive abilities of virtualization to increase performance and efficiency of computer clusters. We target computer clusters due to their reliance on batch schedulers that contain information about the jobs they are running. VIOLIN can use job size and runtime information collected from the batch scheduler to make intelligent adaptation decisions.

5.2 Motivation

To meet the varied computational needs of a large organization it is often necessary to maintain multiple, separate computational domains. These domains are administered independently and will have software, hardware and network environments customized to best serve their organizational unit. The workloads assigned to

these clusters will also vary; while one cluster is experiencing a spike in workload, another may be sitting idle. Clearly this is wasteful of computational resources.

This wastage could be decreased were the organization able to temporarily transfer resources from an under-utilized domain to a busy one. Once the period of peak activity has ended, these nodes could be returned to the original domain. we argue that the throughput of each cluster in an organization could be improved by using virtualization to enable the borrowing of resources during non-overlapping periods of heavy usage

Realizing this goal, however, is not a simple task. Each domain will be configured according to the requirements of its owners. As a result nodes from different domains may not be able to inter-operate. Machines under different domains may have different software packages or user permissions. Worse, one domain may have access to hardware unavailable on another, or be on a private subnet to which other machines do not have access.

Further, organizational units may be unwilling to allow potentially unsafe code to run on their machines, particularly under a privileged account. By lending machines to another cluster, the safety and isolation of its own jobs may be threatened. However, without root access, it may be impossible for the borrowing cluster's jobs to run. All of these challenges must me taken into account for virtual environments to operate efficiently and safely.

In this chapter, we present VioCluster [28], a novel architecture which allows dynamic machine trading while avoiding these problems. We use the concept of VIOLIN virtual environments which allow a cluster to dynamically grow and shrink based on resource demand. Under this system, the administrative privileges of both the borrowing and lending clusters are maintained: cluster administrators are able to configure borrowed machines as required, while not granting root privileges to others making use of their nodes.

A VioCluster uses both machine and network virtualization techniques to logically move machines between domains. A borrowed node in a VioCluster take the form of a

virtual machine running on top of a host machine located in another physical domain. This virtual machine remains completely isolated from its host infrastructure, fulfilling the requirement that administrative access remains exclusively with the node owner. The configuration of the virtual machine is determined by the administrator of the borrowing domain, allowing for the ability to install software packages or hardware as required.

As noted before, users do not generally wish to be encumbered by platform configuration details. To users and applications, the process of borrowing nodes is transparent. A virtual machines running as part of a VioCluster is practically indistinguishable from a physical machine running inside the same domain.

The VioCluster system offers several contributions:

- Dynamic machine trading between mutually isolated virtual environments. VioCluster creates software-based network components which seamlessly connect physical and virtual machines to create isolated virtual clusters. Machines can be traded dynamically through the on-demand creation, deletion, and configuration of virtual machines and network components based on information about the applications gathered from the batch scheduler.

- Dynamic negotiation of machine trades. Each virtual cluster includes a *machine broker* which interacts with other domains. Requests and offers are made through these brokers based on workload and configurable lending and borrowing policies.

We have built a prototype of the VioCluster system, and have demonstrated its effectiveness using two independent Portable Batch System (PBS) [76] based job-execution clusters. Based on job number, size, and duration information collected from the PBS scheduler, environments adaptation decisions are made. Careful application of adaptive measures greatly increase job performance and efficiency. Our performance evaluation results show benefits to both clusters. Most notably they increase their resource utilization and decrease their job execution times.

The remainder of this chapter is organized as follows: Section 5.3 describes the design of VioCluster, Section 5.4 presents key implementation details, Section 5.5 describes the experiments and presents performance results, and Section 5.6 presents the chapter's conclusions.

5.3 Design

There are two key components in the VioCluster system: the ability to create dynamic virtual clusters and the mechanism by which trades are negotiated. This section describes the structure of these components, and the manner in which they interact.

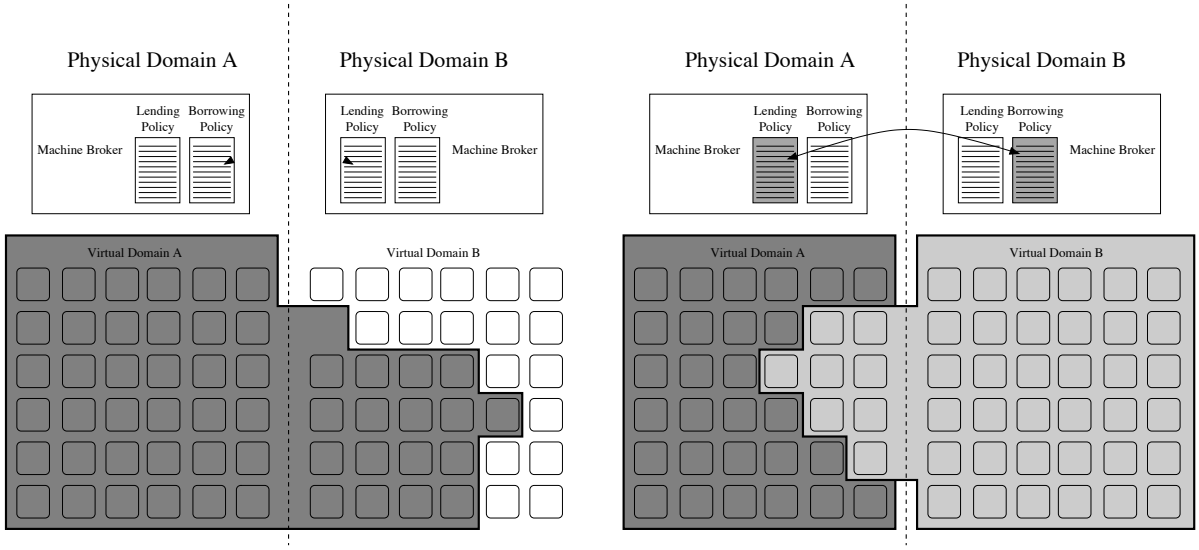
A summary of the terminology used within VioCluster is as follows:

- **Physical domain**: An autonomous set of networked computers managed as a unit. Physical domains have a single administrator, and support a user-base performing specific computational activities. For example, a physical domain belonging to a biology department may be optimally configured for cellular simulations, while a physical domain belonging to a network research group may be designed for shorter network-intensive experiments.

- **Virtual cluster**: An autonomous set composed of a hybrid of virtual and physical machines managed as a unit. Machines in a virtual cluster are connected through a virtual private network, to which both virtual and physical machines have access. Virtual clusters are able to grow and shrink on demand, and they appear to the administrator to be identical to physical domains. A one-to-one mapping exists between physical and virtual clusters; every virtual cluster is hosted upon a physical domain. A virtual cluster is an extension of a physical cluster and utilizes virtual machines on remote domains as if they were local.

- **Machine broker**: A software agent that represents a virtual cluster when negotiating trade agreements with other virtual clusters. A machine broker

consists of a *borrowing policy* which determines under which circumstances it will attempt to obtain more machines, and a *lending policy* which governs when it is willing to let another virtual cluster make use of machines within its physical domain. Both policies are defined by the domain's administrator.



(a) Virtual cluster A borrowing nodes from virtual cluster B in accordance with A's borrowing policy and B's lending policy

(b) Virtual cluster B borrowing nodes from virtual cluster A in accordance with A's lending policy and B's borrowing policy

Fig. 5.1. Two virtual clusters trading resources. Virtual clusters fluidly trade machines in reaction to changes in workload.

Figure 5.1 shows an example of VioCluster consisting of two physical domains, A and B. There are virtual domains associated with each physical domain. Both physical domains consist of 36 machines, each of which initially belongs to its respective virtual cluster. These clusters could be imagined to belong to two university departments, or to two divisions within a company.

Over time, the workload on each virtual cluster varies as jobs are submitted by users. In Figure 5.1(a), virtual cluster A is experiencing a period of heavy demand, while virtual cluster B is under-utilized. After negotiations between the borrowing

and lending policies of the respective brokers, virtual cluster A is able to temporarily borrow half of virtual cluster B's nodes. In Figure 5.1(b), the workload patterns are reversed, and virtual cluster B is able to use nodes located in physical domain A.

When a machine belonging to physical domain B is borrowed by virtual cluster A, it is used to run a virtual machine. This virtual machine is owned by the administrator of virtual cluster A, and will match the configuration of the machines in virtual cluster A. Virtual network connections will be made, connecting the new virtual machine to the nodes of virtual cluster A.

The trading process is authorized according to the borrowing and lending policies within the machine brokers of each domain. These policies are defined by the domain administrator, and allow complete control over the access to a domain's resources. Without an agreement between the brokers, the trade cannot occur.

5.3.1 Adaptive Virtual Clusters

The use of virtualization is key within VioCluster. Through the use of virtual machines, many of the configuration and access problems inherent in machine trading can be avoided. Additionally, virtual networking allows physical and virtual machines to communicate transparently, making network administration no more difficult than for a single physical cluster.

Machine Virtualization

When transferring physical machines between domains without the aid of virtualization, many problems may be encountered. The set of user and group accounts on clusters may be different, leading to access problems when running jobs. Necessary packages and services may not be installed, and superuser permissions may be required to customize a machine's configuration. Additionally, once a borrowed machine is no longer necessary, it must be restored to its original state before it can be used again in its original context.

By using virtualization, VioCluster bypass all of these problems. When a physical node is lent to another virtual cluster, all that is required is a virtual machine process run by an unprivileged user. The virtual machine is created using a *disk image* supplied by the borrowing cluster. The user accounts, services and software services on this image can be configured identically to those on the virtual cluster's physical machines. And when the machine is ready to be returned to its original cluster, all that is required to restore its state is the termination of the virtual machine.

Network Virtualization

Networking between nodes in a virtual cluster is made possible by the use of the VIOLIN networking mechanisms. Traditional VIOLIN creates a virtual layer-2 network overlay that tunnels network traffic end-to-end between remote virtual machines. The overlay appears to these machines to be an isolated physical Ethernet LAN. VioCluster, uses a the core features of the VIOLIN distributed switch, however, it uses a hybrid of virtual machines as well as physical machines in each VIOLIN environment. Within VioCluster, virtual and physical machines are able to exchange network data transparently.

The physical host A connects to the kernel-space virtual NIC, or tun/tap device. Data sent to this device via standard system calls are forwarded to the virtual switch daemons on host B, where they are passed on to the virtual machines. Communication from B to A is symmetrical, with the exception that the virtual NIC on B is located in the kernel space of the virtual, rather than the physical, machine.

The effect of the hybrid approach to using a VIOLIN network overlay is that each virtual cluster has a uniform and private IP address space. Virtual machines running on borrowed nodes can be assigned IP addresses in the same range as the physical machines belonging to the environment. Maintaining this illusion simplifies the administration process and minimizes IP accounting. Additionally, because VI-

OLIN virtualizes addresses at layer-2, arbitrary IP address spaces can be designated to virtual clusters without the threat of conflicts with the host network.

In effect, the hybrid VIOLIN environment extends the LAN. Virtual machines participating in a virtual cluster have the same access to domain services as nodes in a physical domain. For example, all nodes on a virtual cluster may have equal access to NFS-mounted directories and LDAP account information.

The machine and network virtualization techniques used in VioCluster allows the isolation and safety required to make machine trading a viable proposition for large organizations. Fully customizable machine configurations running inside isolated virtual machines and private networks allow virtual clusters that are borrowing nodes the flexibility to perform their work, and ensures that physical domains that are lending nodes are not forced to compromise their security.

5.3.2 Machine Brokering

The second major component of the VioCluster system is the mechanism by which machine exchanges are negotiated. Each domain has a software agent called a *machine broker* which has the responsibility of determining whether trades should occur. The VioCluster system supplies the means by which physical domain administrators may define lending and borrowing policies. Wise decisions made in these policies can lead to large benefits in overall system throughput, while poor choices can degrade performance.

The policies used by a machine broker must be designed with several factors in mind. Observations of the current and past workload levels in the domain may allow predictions of future demand. For example, a cluster may see predictable spikes in usage during business hours, with less demand at night. In that case, a sensible set of policies might take advantage of this usage pattern and borrow during the day and lend at night. Additionally, knowledge of the applications run on a domain may influence policy decisions. For example, if a cluster is used primarily for short,

network-intensive experiments, borrowing nodes may lead to unacceptable communication overheads.

What follows are several attributes that must be specified in the policies for negotiating machine trades:

- **Reclamation**: A policy must determine when a machine will be returned to its home domain. One reclamation policy may be to lend a machine for a specified lease period. Once this lease period has expired, the node must be returned. Alternately, a lending domain may wait until a remote job has completed, or reserve the right to reclaim machines gradually as its demand increases.

  Another possible reclamation policy would be to lend a machine for an unlimited period, on the understanding that it will be returned when required by the owner. While this offers flexibility to the lending domain, it requires the borrower to be able to recover from the sudden loss of a machine. If the borrowing virtual cluster cannot handle this situation, its borrowing policy should forbid such trades.

- **Machine properties**: The characteristics of the machine to be borrowed will have an impact on the policies of the domains involved in a trade. The machine broker must ensure that any machine lent to another virtual cluster has the processing power, memory and disk space required to run a virtual machine. Since the granularity at which resources are assigned is at the machine level, some properties, such as CPU power and memory, will remain constant. However others, such as available disk space, may change and must be monitored by the machine broker. Machines not capable of running a virtual machine suitable for the tasks required may be rejected by the borrowing machine broker.

- **Location**: For some applications, particularly those with high levels of communication between nodes, the physical location of machines may affect performance. Bandwidth and latency within a virtual environment may be affected by the location of communicating nodes. A borrowing policy should be aware

of the communication requirements of its applications. If little communication is required, it may be acceptable to borrow nodes with high-latency or nodes with low-bandwidth connections to the remainder of the machines. Alternately, if the applications run on a virtual cluster tend to be tightly-coupled with high levels of network traffic, it may be best to wait until nodes can be co-located on a single physical domain.

5.4 Implementation

We have implemented a prototype VioCluster system that uses a domain virtualization mechanism based on UML and VIOLIN networking, which is governed by simple but effective machine brokering policies.

As an example application scenario, our prototype configures virtual environments as clusters managed by a PBS job scheduler. Within each virtual cluster one physical machine is designated the PBS master node and the remainder of the machines are configured as PBS compute nodes. As workload changes, machines are added and removed from the virtual cluster and the PBS master is re-configured to allocate jobs to all machines in the virtual cluster. It is important to note that batch scheduling is *not* the focus of our work and PBS is only used as a sample application.

5.4.1 Domain Virtualization Implementation

User-Mode Linux and VIOLIN environments are well suited for VioCluster's machine and network virtualization techniques because of their user level execution needed for isolation. In addition, VIOLIN's networking abilities that support both virtual and physical machines fit VioCluster's unique needs.

Virtual Network Configuration. In our prototype, we define a virtual cluster by a Hybrid VIOLIN network. The machines connected to each virtual cluster's virtual network are a mixture of the real machines of its physical domain and the virtual machines created on nodes borrowed from other physical domains.

Each physical machine has two NICs: one connected to the physical LAN (*eth0*), and the other to the VIOLIN network (*tap0*). Remote domains access the host via a private non-routable IP address associated with *tap0*. Depending on the configuration of the virtual NIC, virtual machines can have Internet routable IP addresses or private non-routable IP addresses through which they can access the Internet through a NAT router. The routing tables of the physical machine must be aware of the virtual network configuration, enabling traffic destined for the physical machine to be sent to *eth0* while virtual machine traffic is forwarded to *tap0*. For this reason, administrative privileges are needed on the physical domain to create and manage the tun/tap device.

Virtual Machine Configuration. VioCluster dynamically creates and destroys UML virtual machines at the request of the machine brokers. The virtual machines are configured by the borrowing domain to fulfill the needs of its applications. In the prototype system, all virtual environments use a modified Fedora Core I root file system. This includes all libraries, packages and applications (such as MPI and PBS) required to function like the physical machines.

Since the virtual machines are nearly-identical PBS compute nodes, small Copy-On-Write (COW) files can be used to capture the differences in their disk images. These COW files share a common root image and store only the minor differences between images, such as network settings. Upon creation, a COW system image represents an individual virtual machine and can be transferred to any host for instantiation.

The prototype system uses a pool of pre-built COW file systems. However, it would also be possible to create them on-demand.

Virtual Machine Instantiation. The creation of a UML virtual machine requires only user-level access to the host machine. The root file system must be transferred, the virtual switch daemon started, and the virtual machine booted. To this end, each physical domain maintains a user account for each peer virtual cluster that may borrow machines.

The size of the root filesystem used in the prototype is approximately 300MB when compressed. Transferring this across the network whenever a virtual machine is required would be prohibitive, so the base system is copied to each potential host before any virtual machine is created. As a result, only the small (approximately 200KB) COW file must be transferred on demand.

Once the root image is transferred, the virtual switch daemon is started via an *ssh* connection. The daemon contacts the peer daemons on the other hosts and joins the distributed switch. Finally, the virtual machine is booted via *ssh*, and connects through the distributed virtual switch.

Virtual Machine Removal. Before a virtual machine can be removed from a virtual cluster, it must be halted. This can be achieved by either killing the virtual machine process or by using the *shutdown* command inside the virtual machine. Killing the process is faster, but results in a corrupted COW file. The common root disk image is read-only, however, so it is unaffected.

The state maintained on the virtual machine's COW files is not important, since the virtual machines are used as PBS compute nodes. As such, when we need to create a new virtual machine, we can simply copy the original COW file from the virtual environment administrator. This means that it is not necessary to properly shut down virtual machines, making the halting process faster.

5.4.2 Machine Brokering Implementation

Virtual clusters share resources according to *contracts* agreed upon by their machine brokers. Trades are permitted or denied in accordance with the brokers policies, based on virtual cluster demand levels. Virtual clusters experiencing heavy workloads propose trade offers, while those with spare capacity advertise the capabilities of the machines they have available. The proposal will be accepted only if the trade is acceptable to the policies of both brokers.

Our implementation of VioCluster uses PBS to schedule jobs. It should be noted that our intent is not to study job scheduling; we use PBS simply to gather information about environmental demand and to demonstrate VioCluster's ability to operate in a dynamic environment. Applications running on virtual clusters must be able to handle changes in available machines. PBS, and most job schedulers, can adapt to these changes and are excellent ways to use the dynamic resources of virtual clusters. When a machine is added or removed from the virtual cluster the PBS master daemon is re-configured to reflect the change. One benefit of using a job scheduler is its inherent resilience to node failure. Nodes can be preempted at any time without effecting the correctness of the applications. Jobs will be re-run by the scheduler with the only effect being on performance. To reduce the effect of preemption on performance, the PBS scheduler is aware of the heterogeneity of the virtual cluster and never schedules jobs on a mixture of virtual and physical machines. The application's ability to adapt is particularly important in our case, since virtual machines are created and destroyed on demand.

Demand Heuristic. In general, there is no requirement for how machine brokers calculate demand. For our prototype, we use the PBS scheduler's work queue as a measure of the demand on the domain. Each virtual cluster a uses a PBS scheduler that multiplexes the set of physical machines P_a and the set of virtual machines V_a. The machine broker queries the PBS scheduler for the number and size of jobs j in the queue Q_a. The result is used to assess the current demand d_a, defined as the number of nodes required to satisfy all jobs $j \in Q_a$.

$$d_a = \sum_{j \in Q_a} j_{nodes\_required}$$

Borrowing and Lending. Based on the calculated demand d_a and current number of machines lent $|L_a|$ or borrowed $|V_a|$, the machine broker calculates the number of machines needed n_a.

$$n_a = d_a - [(|P_a| - |L_a|) + |V_a|]$$

The value of n_a determines if it is desirable to lend, borrow, or return previously borrowed machines. If n_a is positive, virtual environment a needs to acquire n_a nodes; if it is negative then virtual cluster a can lend or return $|n_a|$ nodes; and if it is zero virtual cluster a has exactly enough nodes to satisfy its own demand.

The lending policy implemented in our prototype allows up to half of the domain's idle nodes to be borrowed by other clusters. This allows substantial resources to be offered to other domains in quiet periods, while guarding against resource shortages during sudden spikes in demand.

Reclamation Technique. It is sometimes necessary for a virtual environment to recover machines from other clusters, due to sudden increases in demand. In these cases, machines are returned according to the reclamation policy specified in the lending contract.

In our prototype, machines are returned immediately upon request from the lending domain. Any jobs running on these machines will be terminated, and must restart. This recovery process is managed by the PBS scheduler. Clearly, such preemption has a negative impact on the throughput of the system, and steps must be taken to minimize the cost of this interference. When possible, our scheduler runs jobs only on physical machines belonging to the virtual cluster. Additionally, to minimize the overhead of preemption and improve the network locality of a job, the scheduler never schedules jobs on a mix of virtual and physical machines.

As an alternative, a gradual reclamation policy may be implemented. Future work on VioCluster will study policy interactions, most notably in the area of machine reclamation techniques.

5.5 Experiments

In this section we present several experiments that show the feasibility of VioCluster[1]. First we measure several individual VioCluster system properties, then we show

[1]Special thanks goes to Phil MaGachey who coded much of the VioCluster simulator and helped run the simulations.

the results of a large-scale VioCluster simulation based on the prototype's measured behavior, using real workload traces from production clusters.

5.5.1 System Prototype Measurements

For the prototype measurements we used two clusters on the Purdue University campus. One cluster is administered by the nanoHUB and the other cluster is administered by our laboratory in the Computer Science department. The nanoHUB cluster is composed of dual processor 3.06GHz Intel Xeon machines with 2GB of RAM connected by 100Mb/s Ethernet. The Computer Science cluster is composed of 2.6GHz Intel Xeon machines with 2GB RAM connected the 100Mb/s Ethernet. The connection between the clusters is through Purdue's campus network. Although the UML-based experiments provided good results, our continuing work with Xen promises significantly increased performance.

Metric	Value
Execution Slowdown on virtual machine	15%
Virtual Machine Boot Time	40 seconds
Virtual Machine Halt Time	16 seconds
VIOLIN Bandwidth Penalty	5-15%
VIOLIN Latency Penalty	5-10%

Bandwidth and Latency. From our previous work [15] we have observed that VIOLIN networks affect communication bandwidth and latency. VIOLIN decreases the bandwidth between machines by 5-15% and increases the latency by 5-10%.

Computation Overhead. We have found that computation and communication intensive workloads, such as High Performance Linpack (HPL), run 15% slower on virtual machines connected to a VIOLIN network [19]. Less communication intensive applications would experience a smaller decrease in performance.

Image Transfer, Boot, and Halt Times. Aside from runtime overhead, virtual machines require time to be setup and destroyed. By using small COW filesystems and transferring the base system images ahead of time, we can transfer the system image to a physical host in under a second. This short transfer time means that the virtual machine boot time dominates the creation process. We measured boot times for two virtual machine images: the first being a modified Fedora Core 1 server installation, and the second a minimal RedHat 8.0 system. The larger image took 40 seconds to boot on the our cluster, while the smaller took 5 seconds. Halt times were found to be 16 seconds and 3 seconds respectively.

5.5.2 Simulation

Simulation Setup. Evaluating VioCluster on real workloads was not practical, due to the difficulty of scaling workloads that originally took months or years to run in a reasonable period. We therefore developed a simulator that not only enables us to accurately evaluate the system on large workload traces, but also to simulate far larger clusters than we have available.

Our simulation takes into account our measured virtual machine transfer, boot and halt times, as well as the computation and communication overheads of VIOLIN. The machine brokers use the preemptive trading policy described in section 5.4.2. Each broker calculates the virtual cluster's demand hourly, and then takes action based on the results.

The virtual cluster's size and usage pattern is created using traces obtained from production machines. The two traces used are publicly available (CTC and OSC from [96]) and are composed of a 512 CPU machine at The Cornell Theory Center (CTC) and a 178 CPU machine at Ohio Supercomputing Center (OSC). What follows is the results of the simulation. By using our machine trading mechanism and policy the perceived processing power of both clusters is improved.

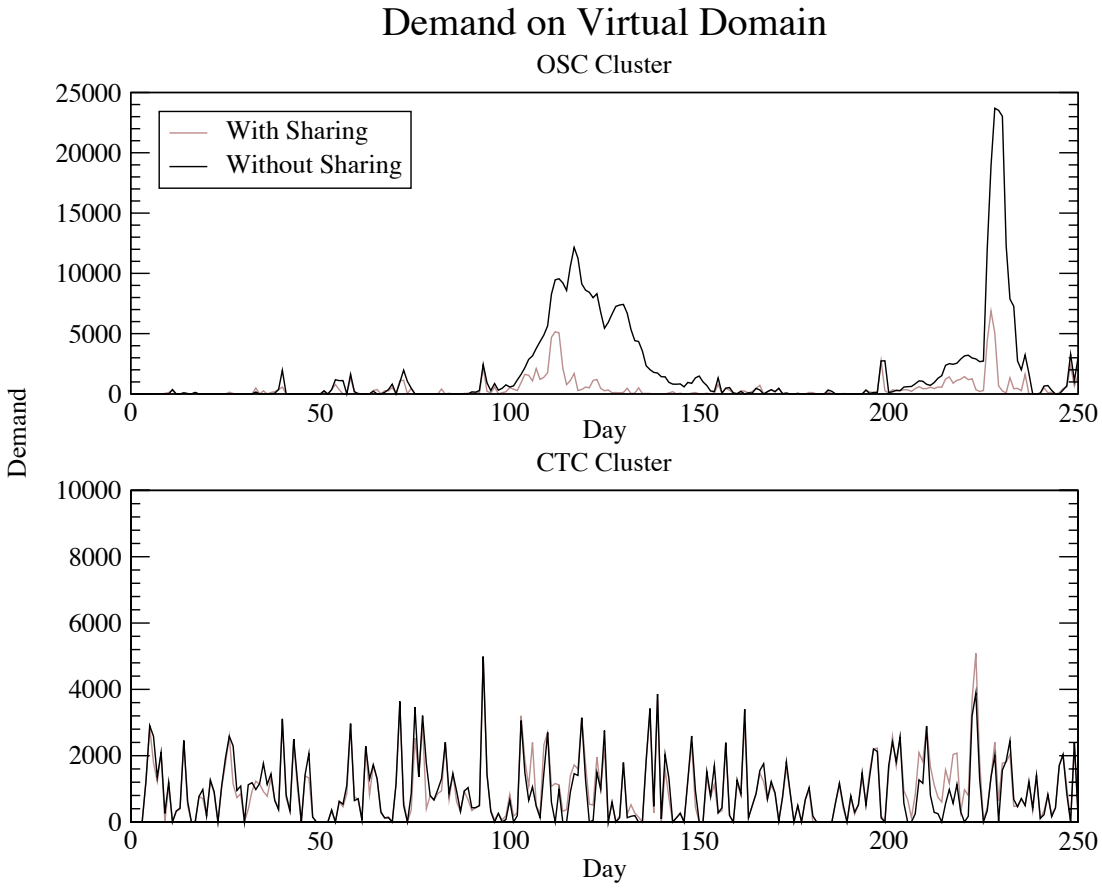


Fig. 5.2. Demand on the virtual cluster over time.

Observed Demand. Figure 5.2 shows the demand on each of two virtual clusters over time with and without sharing enabled. The demand without sharing is what would be observed if the load patterns were submitted to independent clusters. It can be seen that the OSC cluster has two distinct spikes of very high demand and the CTC cluster has many spikes that are relatively small.

When the clusters are run with sharing enabled the virtual clusters are able to handle demand more efficiently. With sharing, the large spikes of the OSC cluster are significantly reduced while the CTC cluster is relatively unaffected.

Machine Borrowing. Figure 5.3 shows the machines traded between domains over time. Positive numbers indicate that the OSC virtual cluster is borrowing from CTC, while negative numbers indicate CTC borrowing from OSC. A correspondence

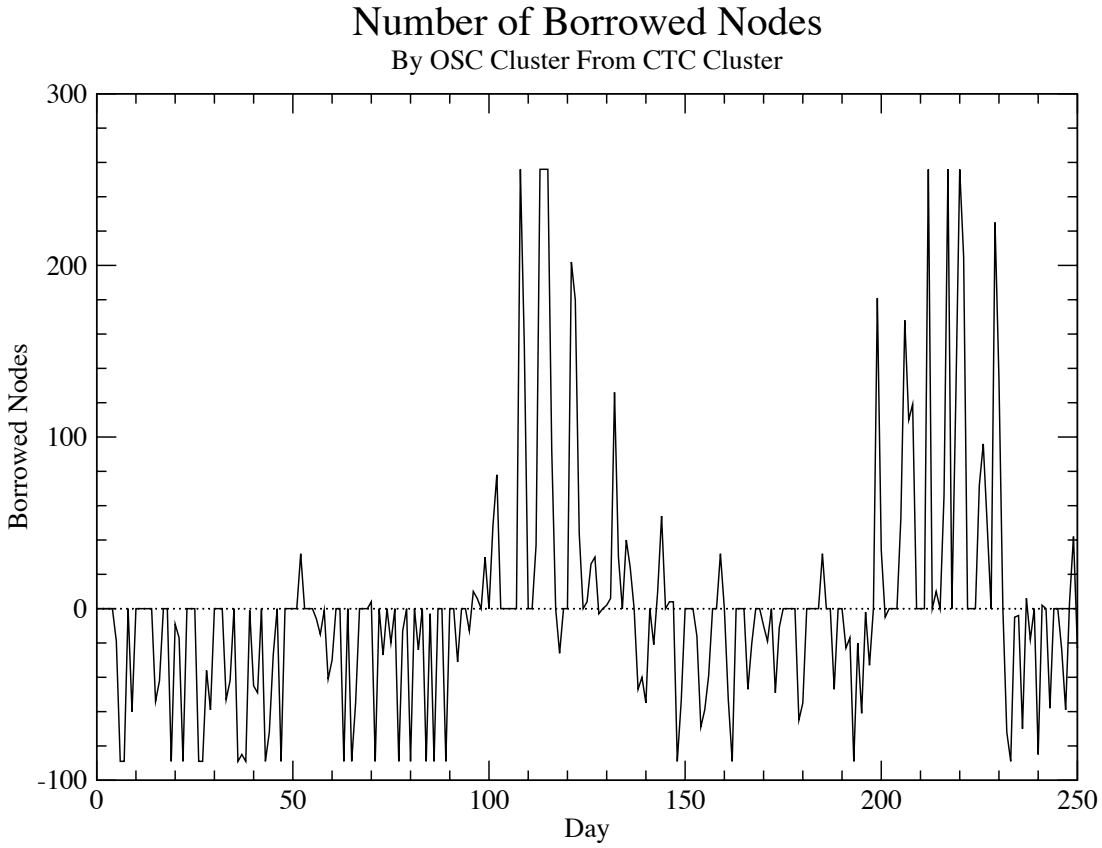


Fig. 5.3. Number of borrowed nodes.

can be seen between the areas of high demand in the OSC domain in Figure 5.2 and the number of machines borrowed from CTC. The high frequency of spikes in Figure 5.3 could be reduced by either a more conservative lending strategy or a more gradual reclamation policy.

Job Completion Times. Figures 5.4, 5.5, and 5.6 show the responsiveness of the system from a user's perspective. They show the average time between a job's submission to the cluster's queue and its completion both with and without sharing. Each figure depicts the interaction of cluster workloads with different qualitative properties.

Figure 5.4 shows the interaction between two different workload patterns. The OSC cluster has two large spikes of high demand, while the CTC cluster has a steady

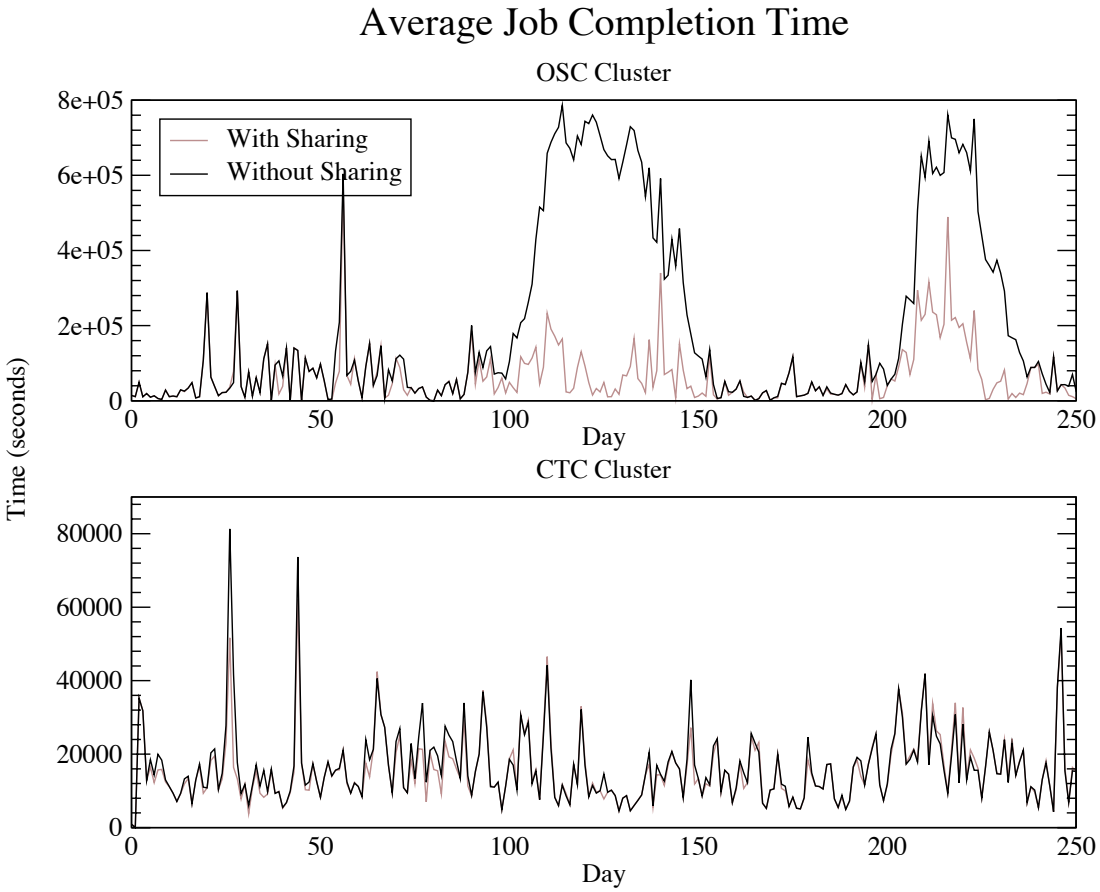


Fig. 5.4. Average time from job submission to job completion of qualitatively different workloads.

pattern of small spikes. From the graph, we see that both clusters benefit from sharing, with the OSC cluster almost completely eliminating its spikes.

The reduction in completion time during peak demand is due to a reduction of the amount of time the jobs wait in the queue. Referring back to Figure 5.3, we see that during periods of high workload, the OSC virtual cluster is often able to borrow a significant portion of the CTC virtual cluster's nodes. The borrowed nodes allow the OSC virtual cluster to run more jobs at once reducing the average completion time drastically.

Figure 5.4 shows occasional points where completion time increases when sharing is enabled. This is caused by jobs being preempted when a physical machine was

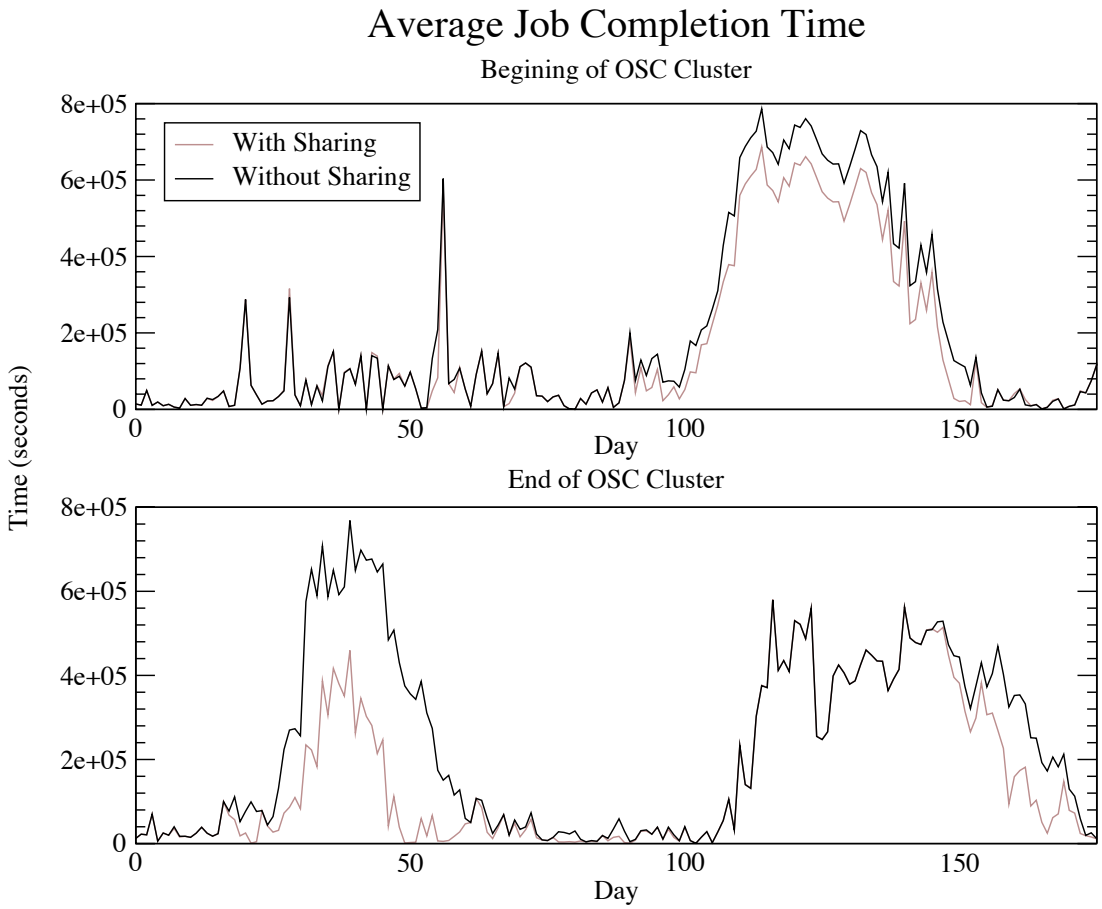


Fig. 5.5. Average time from job submission to job completion of high demand workloads.

reclaimed, and subsequently restarted. Techniques such as check-pointing, virtual machine migration, or complementary advanced scheduling algorithms designed for dynamic systems, can reduce or eliminate this problem.

In Figure 5.5, two traces with corresponding spikes in demand were created by extracting sections of the OSC cluster trace. From the figure we see that if one of the virtual environments has a spike at the same time the other does not, the spike is largely mitigated through machine borrowing. On the other hand, if both virtual clusters experience a spike at the same time, very little improvement is seen, although, neither cluster experiences a reduction in performance.

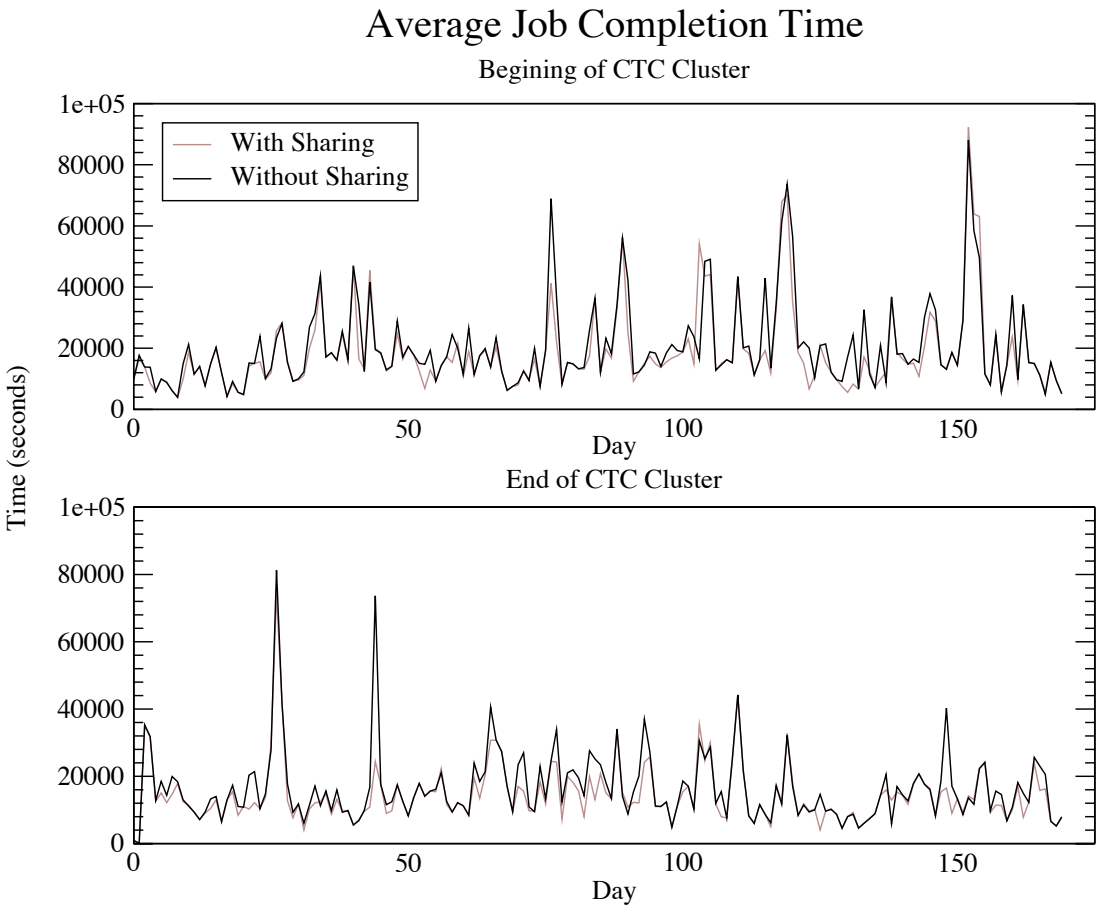


Fig. 5.6. Average time from job submission to job completion of low demand workloads.

Figure 5.6, created from sections of the CTC trace, shows the opposite situation. Here neither cluster experiences a large spike in demand, and as a result, neither benefits greatly from sharing. However, again, neither shows a performance decrease.

From theses results we can conclude that by sharing computational resources through VioCluster great gains in the user's perceived performance are possible. As expected, clusters that are experiencing extreme spikes in workload benefit most from sharing. Unexpectedly, their gains do not penalize the donating clusters. In addition, these gains are the product of a relatively simple trading policy which demonstrates the the effectiveness of the system and the potential of more advanced policies.

5.6 Conclusion

We have presented the design and implementation of VioCluster, a virtualization based computational resource sharing platform based on VIOLIN. Using VioCluster, independently administered computation domains can lend and borrow nodes, increase utilization, and reduce idle node time. We have implemented a prototype VioCluster system and have created a large scale simulation based on real prototype parameters. The results of the simulation using real workload traces show that by using relatively simple machine trading policies VioCluster leads to potentially large increases in the perceived computational power of administratively autonomous clusters. VioCluster proves that the VIOLIN middleware can enable computer clusters to provide better performance to their users.

6. AUTONOMIC VIRTUAL ENVIRONMENTS FOR SHARED CYBERINFRASTRUCTURE

6.1 Introduction

We have seen the emergence of shared distributed cyberinfrastructures that federate, allocate, and manage heterogeneous resources across multiple network domains. The growth of these infrastructures has led to the availability of unprecedented computational power to a large community of users. Meanwhile, virtual machine technology [6, 7, 9] has been increasingly adopted on top of such shared physical infrastructures [11], and has greatly elevated customization, isolation, and administrator privilege for users running applications inside individual virtual machines.

The previous chapters have proposed techniques that enable the creation of virtual distributed computation environments on top of a shared distributed infrastructure and take the initial steps toward dynamic virtual environments. Thus far we have discussed VIOLIN environments and how they are composed of virtual machines connected by a virtual network, which provides a layer separating the ownership, configuration, and administration of the VIOLIN environment from those of the underlying infrastructure. The goal of this dissertation is to create mutually isolated autonomic VIOLIN environments that can be created for different users as their "own" private distributed computation environment bearing the same look and feel of customized physical environments with administrative privilege (e.g., their own private cluster). Within VIOLIN, the user is able to execute and interact with unmodified parallel/distributed applications, and can expect strong confinement of potentially untrusted applications, giving them easy to use access to the vast cyberinfrastructure.

It is possible to realize VIOLIN environments as integrated, autonomic entities that dynamically adapt and relocate themselves for better performance of the applications running inside. This all software virtualization of distributed computation environments presents a unique opportunity to advance the possibilities of autonomic computing [97, 98]. The autonomic adaptation of virtual computation environments is driven by two main factors: (1) the dynamic, heterogeneous availability of infrastructure resources and (2) the dynamic resource needs of the applications running inside VIOLIN environments. Dynamic resource availability may cause the VIOLIN environment to relocate its virtual machines to new physical hosts when current physical hosts experience increased workloads. At the same time, dynamic applications may require different amounts of resources throughout their execution. The changing requirements can trigger the VIOLIN environment to adapt its resource capacity in response to the application's needs. Furthermore, the autonomic adaptation (including relocation) of the virtual computation environment is *transparent* to the application and the user, giving the latter the illusion of a well-provisioned, private, networked run-time environment.

To realize the vision of autonomic virtual environments we must address significant challenges. First, we must provide the mechanisms for application-transparent virtual environment adaptation. In order to provide a consistent environment, adaptation must occur without affecting the application or the user. Currently, work has been done to enable resource reallocation and migration within a local-area network [24] and most current machine virtualization platforms support migration. However, we still need to determine how to migrate virtual machines across a multi-domain environment without affecting the application. The solution must keep the virtual machine alive throughout the migration. Computation must continue and network connections must remain open. The necessary cross-domain migration facility requires two features not yet provided by current virtualization techniques. First, virtual machines need to retain the same IP addresses and remain accessible through the network when physical routers will not know where they were migrated. Second, cross-domain

migration cannot rely on NFS to maintain a consistent view of the large virtual machine image files. These files must be transferred quickly across the network. Clearly, current solutions are not yet adequate for multi-domain infrastructures. VIOLIN virtual environments extend the local network across domain boundaries avoiding these limitations.

The second challenge is to define *allocation policies*. Our goal is to move beyond the limits of static allocation and provide autonomic environments that have the intelligence to scale resource allocations without user intervention. As such, we need to determine when a virtual machine needs more CPU, which virtual machine should be migrated, and where to migrate the virtual machine when a host can no longer support the memory demands of its guests. Consequently, we need to determine if the best destination is that which allows for quick migration, or that which, irrespective of migration speed, can ensure adequate resources.

The main contribution of this chapter is the introduction and analysis of autonomic adaptation capabilities of VIOLIN environments. These environments retain the customization and isolation properties of existing static VIOLIN environments, however, they may be migrated to another host domain during run-time. In this way we can make efficient use of available resources across multiple domains.

We have built a prototype adaptive VIOLIN system using Xen virtual machines and have deployed it over the nanoHUB infrastructure. The evaluation of the system shows that we are able to provide increased performance to several concurrently running virtual environments. To the best of our knowledge, this is the first demonstration of an autonomic adaptive virtual environment, using live application-transparent migration with real-world parallel applications.

6.2 Autonomic Virtual Environments

In the VIOLIN system, each user is presented with an isolated virtual computation environment of virtual machines connected by a virtual network. From the user's

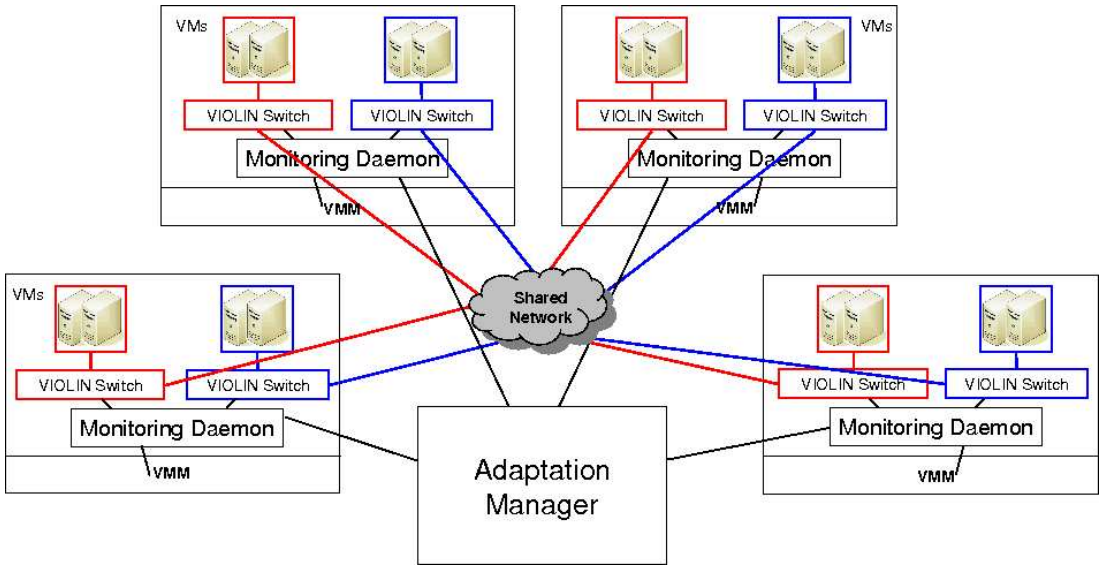


Fig. 6.1. Two VIOLIN environments sharing multiple hosts. Daemons on each host assist the Adaptation Manager in monitoring and controlling resource allocation.

point of view, a virtual computation environment is a private cluster of machines dedicated to that user. The user does not know where the virtual machines reside. On the other hand, the infrastructure sees the environments as dynamic entities that can move through the infrastructure utilizing as much or as little resources as needed.

The components of the VIOLIN system can be seen in Figure 6.1 and are described below:

- **Enabling Mechanisms**: The enabling mechanisms include the VIOLIN virtual environments as well as the *monitoring daemon* running on each physical host. The VIOLIN environments provide an interface to the user and applications, while the *monitoring daemons* monitor the CPU and memory on each host by querying the local *virtual machine monitor* (VMM) for resource availability and utilization levels. In addition, the monitors can manipulate the allocation of resources to local virtual machines.

- **Adaptation Manager**: The *adaptation manager* queries the *monitoring daemons* to form a global view of all host resources available as well as the utilization level of the allocated resources. With this information, the *adaptation manager* can dictate resource reallocation including fine-grained per-host CPU and memory adjustments, as well as coarse-grained migration of virtual machines or whole virtual environments without any user or administrator involvement.

6.2.1 Enabling Mechanisms

Local Adaptation Mechanism. The *adaptation manager* controls all virtual machines through the *monitoring daemons*. VIOLIN environments use both memory ballooning and weighted CPU scheduling to achieve fine-grained control over per-host memory and CPU allocation. Both VMware [9] and Xen [6] enable memory *ballooning* which allows the VMM to change the amount of memory allocated to each virtual machine while the machine is running. At run-time, the *adaptation manager* may decide to modify the memory footprint and percentage of CPU allocated through the monitoring daemons.

Multi-domain Adaptation Mechanism. A key contribution of VIOLIN to autonomic adaptation is the ability to reallocate resources to virtual machines by migrating them live across networks. Live virtual machine migration is the transfer of a virtual machine from one host to another without pausing the virtual machine or checkpointing the applications running within the virtual machine. One of the major challenges of live migration is maintaining any network connections the virtual machine may have open. Modern machine virtualization mechanisms provide live virtual machine migration within layer-2 networks [24]. VIOLIN lifts this limitation by creating a virtual layer-2 network that tunnels network traffic end-to-end between remote virtual machines. The virtual network appears to be an isolated physical Ethernet LAN through which migration is possible. As the virtual machines move through the infrastructure, they will remain connected to their original virtual network.

6.2.2 Adaptation Manager

The *adaptation manager* is the intelligent agent, or "puppeteer" acting on behalf of the users and administrators and making autonomic reallocation decisions. It is appointed two tasks: to compile a global system-view of the available resources and to use this view to transparently adapt the allocation of global resources to virtual environments.

Infrastructure Resource Monitoring

The *adaptation manager* monitors the entire infrastructure by querying the *monitoring daemons* on each host. Via the monitors, it maintains knowledge of all available hosts in addition to the demands of applications running within the VIOLIN environments. Over time both the resources available in the shared infrastructure and the VIOLIN environment's utilization of resources will change. Hosts may be added or removed and VIOLIN environments can be created, destroyed, or enter periods of high or low CPU, memory, or network usage.

Resource Reallocation Policy

The *adaptation manager's* reallocation policy is based on observed host resource availability and virtual machine resource utilization. It uses a heuristic that aims to dynamically migrate overloaded virtual machines between hosts within each domain and, if that is not possible, migrate overloaded VIOLIN environments between domains in the infrastructure. We do not attempt to find the optimal allocation of resources to virtual machines. Instead, we aim at incrementally increasing the performance of the system while minimizing the number of virtual machine migrations and the resulting overhead.

Intuitively, the policy determines a desired resource level for each virtual machine and attempts to assign that amount of resources to a virtual machine. If adequate

resources cannot be obtained locally, the virtual machine may be migrated to another host or the whole VIOLIN environment may be migrated to another domain.

It may be that there are not enough resources in the entire infrastructure to supply each virtual machine with its *desired resource level*. In this case, we would like to achieve a weighted balance of load on each domain and host (more powerful hosts/domains will take on more load). Conveniently, a weighted balance of load on an under-utilized system will assure that all (or most) virtual machines will have been allocated their *desired resource level*. With this in mind, our reallocation policy is designed to balance the load between hosts and domains.

The *desired resource level* of each virtual machine is determined by the amount of allocated CPU and memory as well as the amount of resources that are actually being utilized. We wish to keep each virtual machine's resource utilization within a certain (predefined and configurable) range. A utilization level outside of the expected range will cause the *adaptation manager* to increase or decrease the virtual machine's resource allocation.

The heuristic finds over- and under-utilized virtual machines and attempts to adjust their allocations using first the local host's resources. If the local host cannot support all of its currently hosted virtual machines, an attempt is made to find another host within the domain to which one or more virtual machines can be migrated. The heuristic first looks at the hosts within the domain that have the lowest utilization level. If no host can support the over-utilized virtual machine, the whole domain is considered overloaded, and an attempt is made to find another domain which can support the resource needs of one or more of the overloaded domain's VIOLIN environments. If a destination domain is found, VIOLIN environments will be migrated live to hosts in that domain.

Resource Demand Prediction

A fundamental part of the adaption manager's responsibilities rely on its ability to predict the amount of resources each virtual environment and virtual machine will need to utilize in the near-future. Predicting the future is difficult in general and is made even tougher in this scenario because we have no knowledge of the applications that are creating the demand. In light of this restriction, the adaptation managers use only historic data to predict future resource demand.

The goal is to assign a *desired resource level* for each virtual machine in the system. In order to find this level, we use weighted moving average time series analysis [99]. For each virtual machine we observe and record over time a of series of a measurements of the amount of each resource that is utilized. For each resource r, we use the previous n recorded measurements at any given point in time m. The recorded resource utilization measurements form the series $r_m, r_{m-1}, ..., r_{m-n+2}, r_{m-n+1}$. The *predicted usage* is defined as:

$$PU_m = \frac{nr_m + (n-1)r_{m-1} + ... + 2r_{m-n+2} + r_{m-n+1}}{n + (n-1) + ... + 2 + 1} \tag{6.1}$$

The predicted usage gives a prediction of the amount of a resource that a given virtual machine will use in the next time step. However, this prediction is only accurate if the virtual machine is using less than its allocated amount of resources. For example, if a virtual machine has been allocated 10 units of CPU and is only using 4 units, we know that the virtual machine can utilize exactly 4 units of CPU. However, if the same virtual machine is using all 10 units of CPU we can not know how much CPU it may use in the future. We only know that it can use at least 10 units.

We do not directly apply *predicted usage* to *desired resource level*. Instead, we over-allocate resources to virtual machines by a user-defined factor α. For most virtual machines *desired resource level* is defined to be:

$$DRL_m = PU_m * \alpha \tag{6.2}$$

We recognize resource utilizations that are at or near 100% have a greater chance of being able to utilize far more resources than that $PU_m * \alpha$. To address this, we provide a second function used to determine the *desired resource level* of virtual machines using high percentages of their allocated resources. We first define β to be the desired over-allocation of a 100% utilized virtual machines. Then we augment our definition of *desired resource level* to be:

$$DRL_m = PU_m * (\frac{\beta - \alpha}{1 - \alpha} * (PU_m - 1) + \beta) \tag{6.3}$$

Having two functions, one for high utilizations and one for low utilizations could result in small changes in utilization leading to large changes in predicted demand. For this reason, the resulting functions must be continuous at $PU_m = \alpha$. Combining the continuous Equations 6.2 and 6.3 the desired resource level is defined to be:

$$DRL_m = \begin{cases} PU_m * \alpha & \text{if } PU_m < \frac{1}{\alpha}. \\ PU_m * (\frac{\beta - \alpha}{1 - \alpha} * (PU_m - 1) + \beta) & \text{if } PU_m \geq \frac{1}{\alpha}. \end{cases} \tag{6.4}$$

As an example, if we desire to have each virtual machine utilizing 75% of its allocation and a 100% utilized virtual machine to desire twice its currently allocated resources, we would set $\alpha = 1/0.75$ and $\beta = 2$. In the example, the allocation of resources would be done using the *desired resource levels* determined by the function depicted Equation 6.4. All virtual machines with utilizations less than 75% are allocated resources such that their utilization will be 75% while all virtual machines with allocations grater than 75% are assigned allocations along the linear function between 75% and 200%.

Its is important that this desired resource level be assigned such that small changes in utilization do not create large changes in desired resource level. For example, if

our functions were not equal at utilizations of $1/\alpha$, a small increase in utilization that moves the utilization from less than $1/\alpha$ to greater than $1/\alpha$ would create larger changes in the desired resource level. In turn, small oscillations of utilization from less than $1/\alpha$ to greater than $1/\alpha$ would create large oscillations in desired resource level potentially causing larger oscillations in environment adaptation.

Adaptation Model

The model used to develop adaptation policies is based on knowledge gathered from the host-level monitoring daemons. At any given time the adaptation model knows two things: (1) the current mapping of virtual machines to hosts, and (2) the *desired resource level* of each virtual machine as calculated using the method in the previous section. With this knowledge the adaptation manager is tasked with finding the best allocation of resources to virtual machines.

We assume that we are given a set of domains $D = (d_1, d_2, ..., d_n)$ with capacities c_i, a set of virtual environments $V = (v_1, v_2, ..., v_m)$ with desired resource level r_j, and a current mapping of virtual environments to domains $M(v_i) \rightarrow d_j$.

For each virtual machine we define *TargetUtilization* and *DomainUtilization* to be:

$$TargetUtilization = \frac{\sum_{i=1}^{m} r_i}{\sum_{j=1}^{n} c_j} \tag{6.5}$$

The *TargetUtilization* is the ratio of the total demand on the entire system to the capacity of the system. A *TargetUtilization* that is less than one (1) occurs when the system has excess capacity while greater than one (1) implies an over allocated system.

Similarly, the *DomainUtilization* is the ratio of the demand on a specific domain to the capacity of that domain.

$$DomainUtilization(j) = \frac{\sum_{M(v_i)=d_j} r_i}{c_j} \tag{6.6}$$

Next we define the *balance* of a mapping to be:

$$Balance(M) = \sum_{i=1}^{m} |DomainUtilization(M(v_i)) - TargetUtilization()| \qquad (6.7)$$

Observe that if the *DomainUtilization* of all domains were equal, the load on the system would be balanced equally throughout the system. Also, observe that in order for all *DomainUtilizations* to be equal to each other, they would equal the *TargetUtilization*.

These observations lead to the following problem definition:

$$\begin{gathered} \text{Find a new mapping } M'(v_i) \rightarrow d_j \\ \text{Minimizing } Balance(M') \end{gathered} \qquad (6.8)$$

In effect, this model aims to balance the load between domains by minimizing the difference between the utilization of each domain and the utilization of the entire system.

Adaptation Heuristic

We have developed a heuristic based on the model in the previous section. The goal of the heuristic is to balance the load on resources while maintaining the expected performance of the individual virtual environments.

Through our experience with the nanoHUB we have made several observations about the three resources that VIOLIN can control.

- *Networking.* Most cyberinfrastructures and Grids, including the nanoHUB, are composed of several cluster computers. In these environments, there are high-bandwidth, low-latency connections between nodes within an individual cluster and low-bandwidth, high-latency connections between clusters. Further, most applications using these systems, and specifically applications using the

nanoHUB, are tightly coupled parallel applications. To provide better performance to the virtual environments we make the simplifying assumption that virtual environments in the nanoHUB must be completely within a single cluster or host domain. As a result, the execution of any parallel or distributed job will not be required to communicate across domain boundaries.

- *Memory.* We also observe that applications do not perform well if insufficient memory is allocated to them. As such, it is not necessary to load balance the memory allocated to virtual machines. Providing less than adequate memory to an application will, in effect, stop the application due to thrashing. Our heuristic assumes adequate memory allocation. If there is not enough memory in the system to allocate sufficient memory to every virtual machine, some virtual machine will have to be halted. Policy that decides which machines to halt should be decided by the particular needs of the cyberinfrastructure members.

- *CPU.* The processor is the only resource that can be viably load balanced. Jobs not receiving enough processing power will run slower; but for computationally intensive applications, the speed will be proportional to the processor allocation.

With these observations in mind, our heuristic assumes memory to be fully allocated and all virtual environments must be completely within a single host domain. These assumptions have lead us to a two tiered approach to the allocation of virtual machines to hosts. First, virtual environments will be allocated to host domains. Second, each host domain will allocate virtual machines to host machines.

Algorithm 1 depicts the method used to balance load between host domains. The outer loop (line 2) of the algorithm continues the balancing the load while the new mapping (M') provides better balance than the previous mapping (M). During each pass through the loop, the target utilization (line 4) and domain utilization (loop 5) are calculated using the equations seen above. Then we find the domains with the highest and lowest domain utilizations (lines 8 and 9). A virtual environment is

Algorithm 1: Balance load between domains

Input: $D = (d_1, d_2, ..., d_n)$, $V = (v_1, v_2, ..., v_m)$, M = CurrentMapping of $D \rightarrow V$

Output: M = New Mapping of $D \rightarrow V$

1 $M' = M$

2 **repeat**

3 $M = M'$

4 TargetUtilization = $\frac{\sum_{i=1}^{m} r_i}{\sum_{j=1}^{n} c_j}$;

5 **foreach** *Host Domain j* **do**

6 DomainUtilization$_j$ = $\frac{\sum_{M(i)=j} r_i}{c_j}$

7 **end**

8 HighDomain = Max(DomainUtilization)

9 LowDomain = Min(DomainUtilization)

10 MigrationCandidate = $BestFit$(HighDomain,LowDomain)

11 $M' = Migrate$(M,MigrationCandidate,HighDomain $\rightarrow$ LowDomain)

12 **until** $Balance(M') \leq Balance(M)$;

13 **return** M

chosen to be the migration candidate (line 10) and will be migrated from the highest domain utilization to the lowest. The migration candidate is chosen using the *bestfit* heuristic which finds the virtual environment that when migrated will have the most effect on the minimization of the balance function. After the migration candidate is found the new balance is calculated. If the new balance is less than the old one the algorithm continues.

6.3 Implementation

We have implemented an adaptive VIOLIN system prototype and have deployed the system on the nanoHUB's infrastructure. The nanoHUB is a virtualization-based cyberinfrastructure running online and on-demand nanotechnology applications, and is our "living lab". Part of the nanoHUB allows students and researchers to execute computational Nanotechnology applications, including distributed and parallel simulations, through either a web-based GUI or a VNC desktop session. The unique property of the nanoHUB is that the back-end processing is heavily reliant on virtualization. Users of the nanoHUB may, unknowingly, be using VIOLIN environments that have the ability to adapt resource allocation to the changing needs of their simulations.

6.3.1 Deployment Platform (nanoHUB)

The unique property of the nanoHUB is that the back-end processing is heavily reliant on virtualization. Jobs are transparently executed on one of many Grid infrastructures. Unbeknownst to the user, their jobs may be submitted to a local cluster, the TerraGrid, or any Globus or Condor systems. Additionally jobs may be executed on a VIOLIN virtual environment.

VIOLIN virtual environments greatly increase the functionality and efficiency of the nanoHUB. Functionally, VIOLIN environments provide the nanoHUB with the ability to host applications that do not or cannot run on traditional Grid infras-

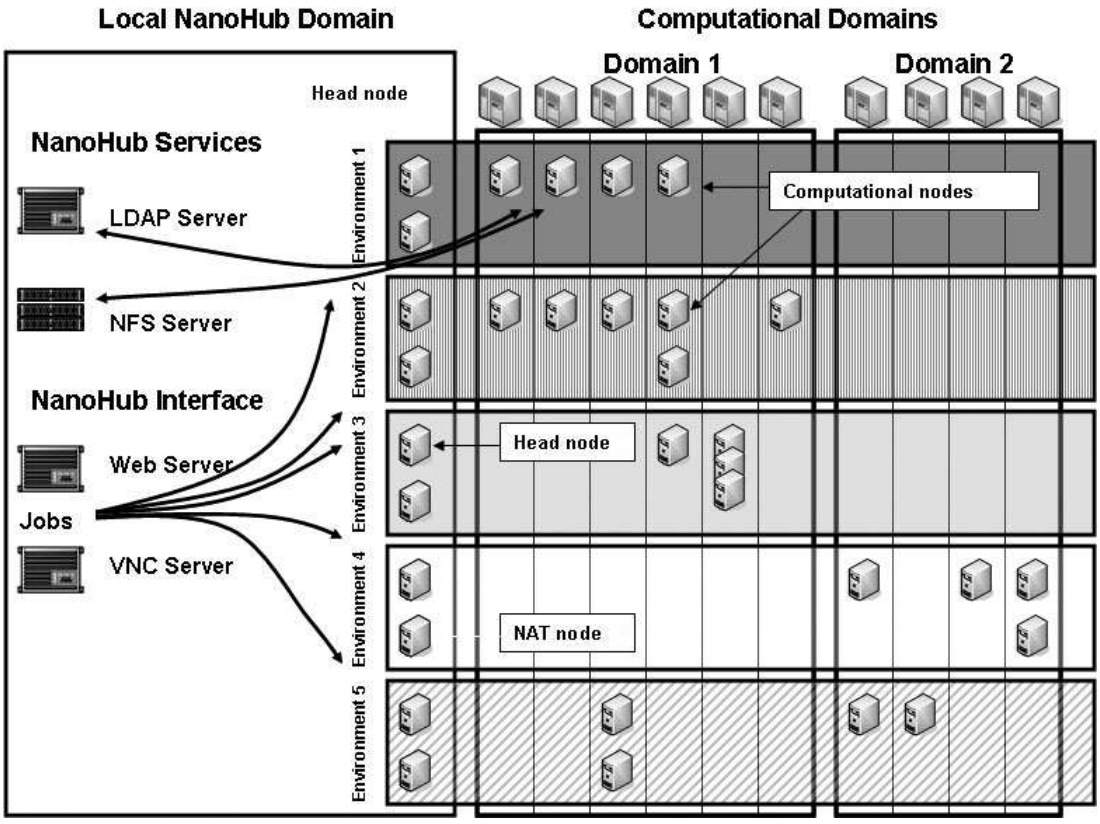


Fig. 6.2. nanoHUB deployment of VIOLIN environments.

tructures. The nanoHUB no longer relies on administrators of remote domains to configure remote hosts and install the necessary specific software packages. Instead, custom virtual machine images can be created to host any existing or future Nanotechnology applications. In addition, it will be possible to allow individual nanoHUB users to own and customize their own virtual environments, removing the need for nanoHUB staff intervention. Further, the efficiency with which nanoHUB applications use the available resources will be increased through the dynamic re-allocation of resources to virtual environments.

6.3.2 Deployment Details

Toward a full deployment, we have deployed multiple adaptive VIOLIN environments on the nanoHUB's multi-domain infrastructure on the campus of Purdue University.

Host Infrastructure. The virtual machines are hosted on two independent clusters on separate subnets. One cluster is composed of 24 Dell 1750s each with 2GB of RAM and two hyper-threaded Pentium 4 processors running at 3.06 GHz, while the other is 22 Dell 1425s each with 2GB of RAM and two hyper-threaded Pentium 4 processors running at 3.00 GHz. Both clusters support Xen 3.0 virtual machines and VIOLIN virtual networking.

Virtual Environment Configuration. Each virtual computation environment is composed of Xen virtual machines connected by a VIOLIN network. Among the virtual machines, one is a head node and the rest are compute nodes. The head node provides users with access to the VIOLIN environment and, as such, must remain statically located within its original host domain. However, all compute nodes are free to move throughout the infrastructure as they remain connected via the VIOLIN virtual network.

User accounts are managed by a shared Lightweight Directory Access Protocol (LDAP) server, and user's home directories are mounted to the local NFS server with the head node acting as a NAT router for the isolated compute nodes, giving a consistent system view to all virtual machines regardless of the physical locations of the virtual machines.

In order to migrate a virtual machine, the following must be transferred to the destination host: a snapshot of the root file system image, a snapshot of the current memory, and the thread of control. Xen's live migration capability supports efficient transfer of the memory and thread of control. It performs an iterative process that reduces the amount of time the virtual machine is unavailable to an approximately 165ms [24]. However, Xen does not support the migration of the root file system

image. Xen assumes that the root file system is available on both the source and destination hosts - usually through NFS which can not safely be made available between multiple domains. The shared infrastructure is composed of independently administered domains which cannot safely share NFS servers.

In order to perform multi-domain migrations, our prototype uses read-only root images that can be distributed without having to be updated. We do this by putting all system files that need to be written to in *tmpfs* filesystems. Since *tmpfs* file systems are resident in memory, Xen will migrate these files with the memory. Initially, we thought of this solution as a workaround to be fixed later. However, our experience has demonstrated that *tmpfs* can be a reasonable solution for a number of nanoHUB applications. In addition to using *tmpfs* for system files, users home directories are NFS-mounted through the virtual network to the nanoHUB server and do not need to be explicitly transferred.

6.4 Experiments

In this section, we present several experiments that show the feasibility of adaptive VIOLIN environments. First, we measure the overhead of live migration of VIOLIN environments, then we demonstrate application performance improvement due to autonomic live adaptation of VIOLIN environments sharing a multi-domain infrastructure. For all experiments we use the nanoHUB VIOLIN deployment, an *adaptation manager* employing the heuristic described in section 6.2.2, and the NEMO3D [95] parallel atomic particle simulation as the application running in the VIOLIN environments.

6.4.1 Migration Overhead

Objective. This experiment aims to find the overhead of migrating an entire VIOLIN environment that is actively running a resource intensive application (individual virtual machine migration overheads have been studied in [24]). The overhead

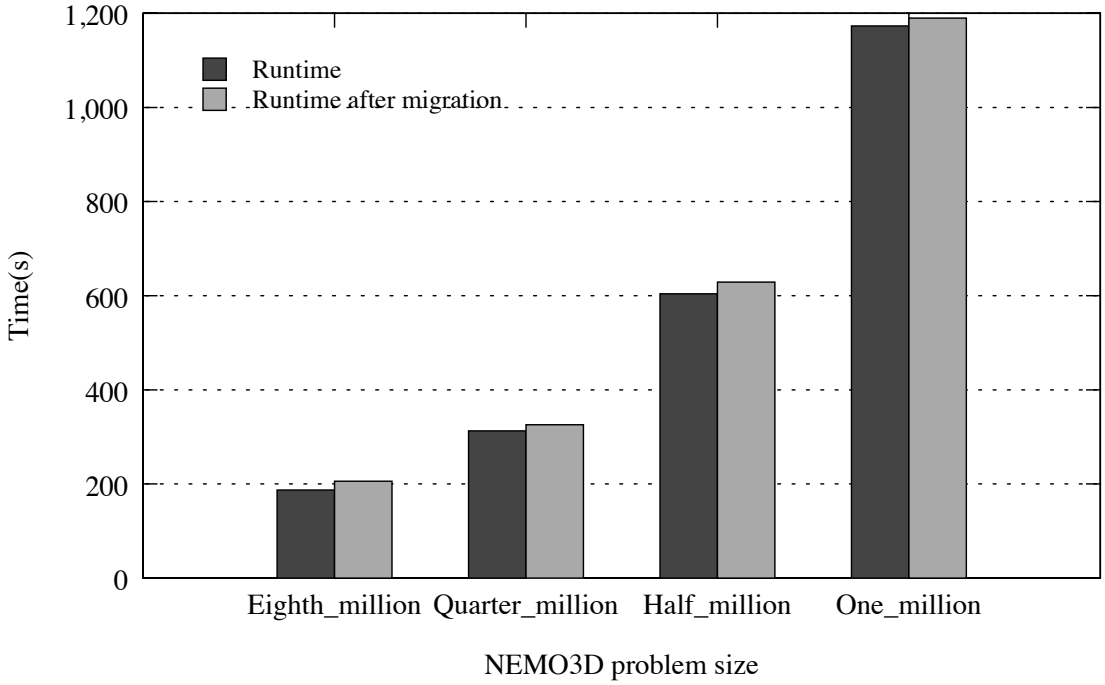


Fig. 6.3. Migration overhead caused by live migration of entire VIOLIN virtual environments that are actively executing the parallel application NEMO3D

of live VIOLIN environment migration includes the execution time lost due to the temporary down-time of the virtual machines during migration, the time needed to reconfigure the VIOLIN virtual network, and any lingering effects such as network slowdown caused by packet loss and the resulting TCP back-off.

Configuration. We use a VIOLIN environment composed of four virtual machines. We execute NEMO3D with several different problem sizes between 1/8 and 1 million particles. For each problem size, we record the execution time with and without migrating the VIOLIN environment. During the no-migration runs, the application is allowed to run unimpeded. During each run involving migration, all four virtual machines are simultaneously migrated live across the network to destination hosts configured identically to the source hosts. In order to stress the system and find the worst overhead possible, we choose the migration to occur at the most resource intensive period of the application's execution. During each run, there is no

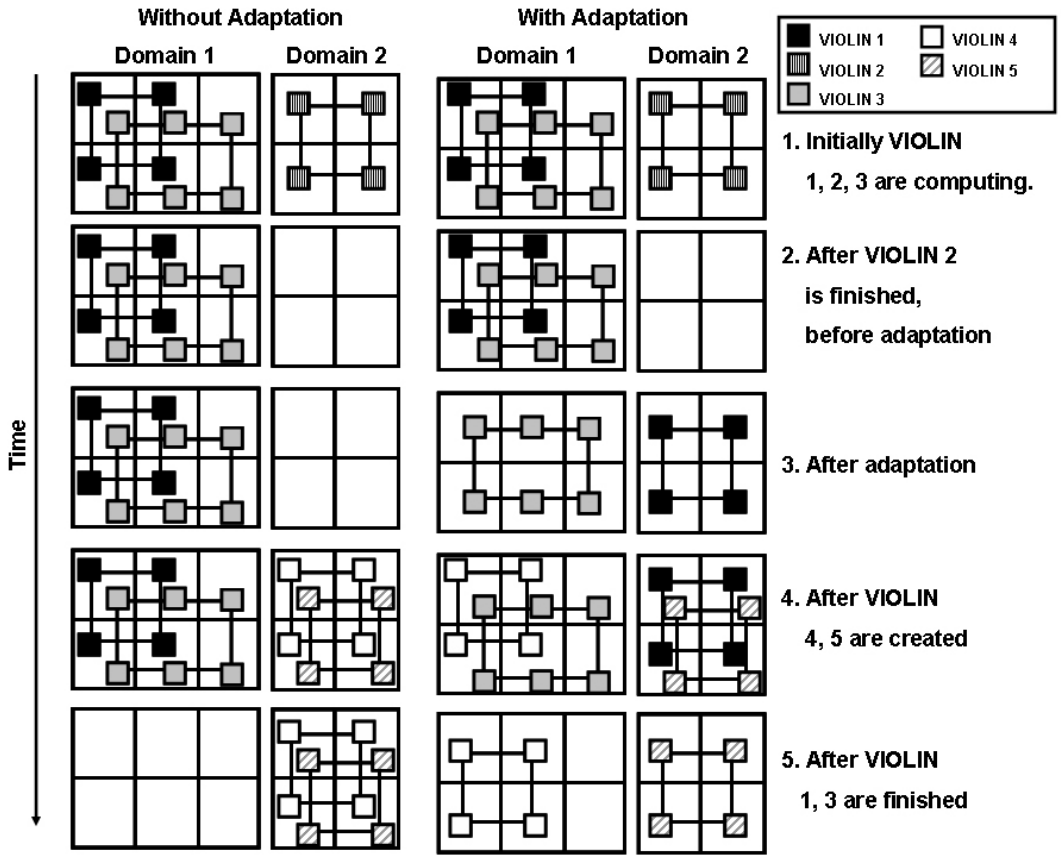


Fig. 6.4. VIOLIN Environment Adaptation Scenario 1.

background load in any of the hosts involved. However, the network is shared and therefore incurs background traffic.

Results. Figure 6.3 shows the results. We find that, regardless of problem size, the run-time of the application is increased by approximately 20 seconds (ranging from 17-25 seconds) when the VIOLIN environment is migrated.

Discussion. One requirement of adaptive VIOLIN environments is that there should be little or no effect on the applications due to adaptation. The 20 second penalty would seem impossible considering that Xen virtual machine migration requires the transfer of the entire memory footprint (approximately 800MB per virtual machine for an execution of NEMO3D simulating 1 million particles). However, Xen's live migration mechanism hides the migration latency by continuing to run the ap-

plication in the virtual machine on the source host while the bulk of the memory is being transferred. We do not measure the actual down-time of our virtual machines; however, Xen migration of a virtual machine with 800MB of memory was found to have a 165ms down-time when migrating within a LAN [24]. The major effect on application performance is not due to the migration itself but the time to reestablish the VIOLIN virtual network plus application slowdown *during* the migration. This experiment shows that the penalty for migrating a VIOLIN environment is relatively small and does not escalate with increased virtual machine memory size.

6.4.2 VIOLIN environment Adaptation Scenario 1

Objective. The purpose of this experiment is to demonstrate the effectiveness of the *adaptation manager* and to show how a small amount of autonomic adaptation can lead to better performance of all VIOLIN environments that share the infrastructure.

Configuration. We launch five VIOLIN environments, each running the NEMO3D application with different input problem sizes (emulating independent VIOLIN environments used by different users). Each VIOLIN environment starts executing the application at a different time. The shared infrastructure is comprised of two host domains. Domain 1 has six physical nodes while domain 2 has four physical nodes. The two domains are subsets of the two physical clusters in the nanoHUB. At the time of this experiment we did not have administrative privileges on any machines outside of Purdue University campus that could be used for these experiments, therefore we did not experiment with truly wide-area infrastructures. However, the two domains that we are using are on separate subnets within Purdue University's campus. These domains have the same routing and migration configurations that would be seen in a true wide-area experiment, and demonstrate VIOLIN's ability to operate in a multi-domain infrastructure.

The experiment compares the execution time of NEMO3D within each VIOLIN environment with and without autonomic resource reallocation enabled. When re-

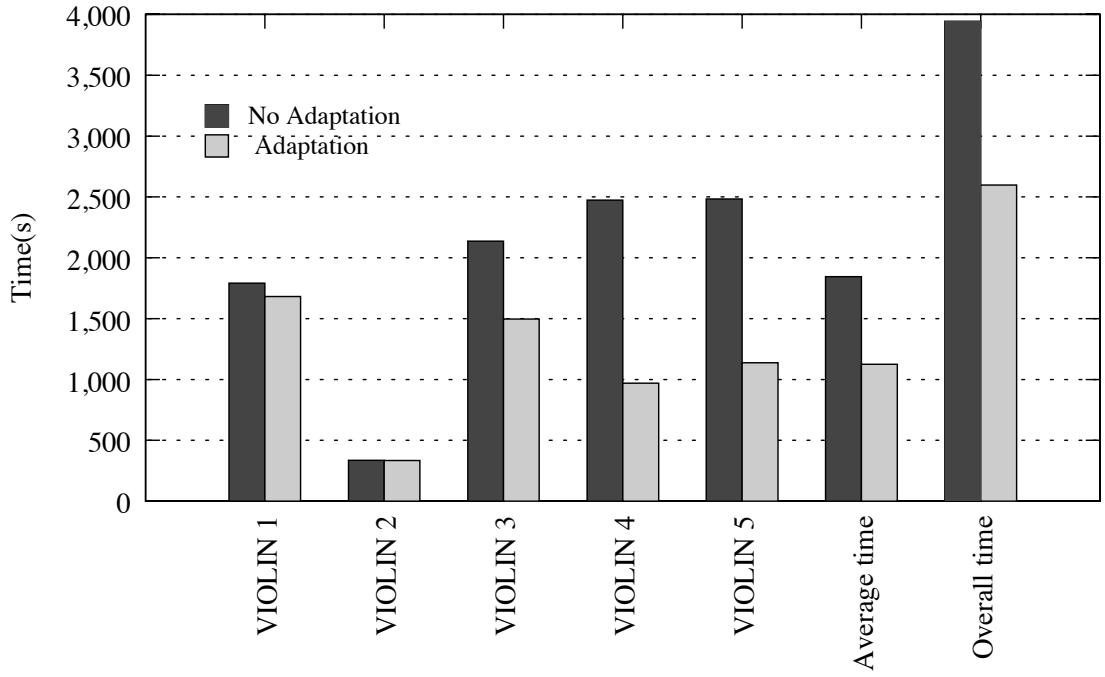


Fig. 6.5. VIOLIN environment Adaptation Scenario 1: Execution time of applications running within VIOLIN environments with and without adaptation enabled.

allocation is enabled, some VIOLIN environments will be migrated in accordance with the *adaptation manager's* heuristic in order to balance the load and improve the performance of applications.

Results. Figure 6.4 is a time-line showing where each VIOLIN environment is located at key instances of time. Figure 6.5 shows recorded NEMO3D execution time of each VIOLIN environment with and without adaptation enabled.

Initially, for both runs, VIOLIN environments 1, 2, and 3 (referred to as V1, V2, and V3) are executing their applications and have been allocated significant portions of the host domains (referred to as D1 and D2). Each virtual machine is using nearly 100% of its allotted CPU.

V2 is executing a smaller problem size and is running alone in D2 so it finishes quickly. When V2's finishes, there occurs a load imbalance between the domains. There are 10 virtual machines in D1 that expect more CPU allocation while there is

no virtual machine in D2. The imbalance triggers the migration of V1 to the hosts of D2. This adaptation balances the load and allows the virtual machines of both V1 and V3 to be allocated the full resources of a single host.

It is important to note that although both remaining VIOLIN environments have increased CPU allocation, V1 temporally slows down during the migration. V3 will surely complete its application sooner, but it remains to be seen if the increased resource allocation to V1 can compensate for the cost of migration.

After some time, V4 and V5 start their applications and require significant resources (100% utilization). We assume that both of these environments are new and must be created to allow the non-adaptation case to have some balance in load. Without this allowance, V4 and V5 would have to remain where they were (potentially within D1, creating an even larger advantage for the adaptation case). In either case, the creation of V4 and V5 causes both domains to be overloaded. The load is, however, balanced.

Next, V1 and V3 finish their applications. From Figure 6.5, we see that the migration of V1 allows V3 to finish 30% sooner than it would have otherwise, while V1 finishes in approximately the same amount of time due to the additional cost it pays to migrate. Once V1 and V3 finish, the remaining VIOLIN environments (V4 and V5) are already balanced in the adaptation case, while they are not in the non-adaptation case.

Table 6.1
Environment Properties

Env	VMs	Size	Start Time
1	4	1/2 Mill Part.	0:00
2	4	1/4 Mill Part.	0:00
3	6	1 Mill Part.	0:00
4	4	1/2 Mill Part.	24:20
5	4	1/2 Mill Part.	24:20

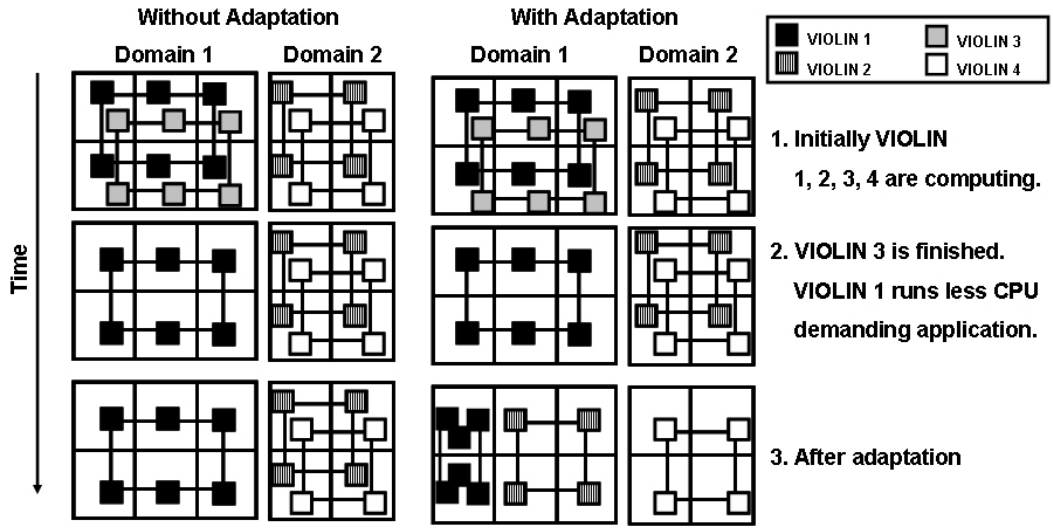


Fig. 6.6. VIOLIN environment Adaptation Scenario 2.

The chart in Figure 6.5 shows the application execution in each VIOLIN environment. For each VIOLIN environment, the execution time is reduced by enabling autonomic adaptation. The last two data points on the chart show the *average time* and *overall time* metrics of the system. The *average time* is the average execution time for all VIOLIN environments. In this example, adaptation saves on the average 39% of the application's execution time. The *overall time* is the duration between the execution of the first VIOLIN environment and the completion of the last VIOLIN environment. The *overall time* gives us a measure of the efficiency of resource usage. We see a 34% reduction in *overall time* with adaptation.

Discussion. Observe that during this experiment nearly all of the VIOLIN environments benefit from adaptation even though only one is migrated, suggesting that a small amount of adaptation can lead to a large increase in both application performance and resource utilization. In addition, heuristics that aim to balance load while minimizing the cost of migration are likely to achieve satisfactory performance without having to find the optimal allocation of resources to virtual machines.

6.4.3 VIOLIN environment Adaptation Scenario 2

Objective. Whereas the previous example shows the typical case where virtual environments are either being heavily used or completely idle, the next example shows how adaptation can benefit applications that go through periods of high and low use during a single execution. In this situation, we create a VIOLIN environment that initially uses a high amount of CPU then moves to a stage in its application that uses lower amounts of CPU.

Configuration. The configuration uses the same host infrastructure as the previous example. However, the VIOLIN environments and their applications have changed. There are now four VIOLIN environments, all of which execute the NEMO3D application except for V1. V1 executes the high demand NEMO3D followed by a less CPU intensive "dummy" application. V1 is simulating 100% utilization followed by a lower utilization that stabilizes within the desired utilization range after the appropriate reduction in CPU allocation.

Results. The time-line in Figure 6.6 and the chart in Figure 6.7 show the resulting execution time of the applications with and without adaptation enabled. Initially, the load is balanced between the four VIOLIN environments which are running on the two domains. After some time, V3 completes its application and no longer requires resources. Next V1 enters its second, less CPU intensive, stage of its execution. In the new stage, V1's utilization of resources drops well below desired range. Its drop in CPU allocation results in a load imbalance between the two domains, forcing the *adaptation manager* to migrate V2 to D1. The migration balances the load between domains but causes an imbalance between the hosts of D1. Since it is now possible for all six virtual machines from V1 to be supported by only two of the available hosts, they are migrated to the hosts left vacant by V2.

The results in Figure 6.7 show that V1 and V2 execute in approximately the same amount of time while V3 and V4 show significantly lower execution time. With

autonomic reallocation enabled, the *average time* and *overall time* are decreased by 41% and 47%, respectively.

Discussion. From this experiment we see that it is possible to obtain further improvement of performance and efficiency by combining the fine-grained resource reallocation mechanisms with the coarse-grained migration mechanisms. The adaptation manager is able to identify virtual environments that experience a significant reduction in resource requirements. By scaling down the CPU share allocated to individual virtual machines of V1, it opens the possibility of migrating V2 thus improving the performance seen by all VIOLIN environments.

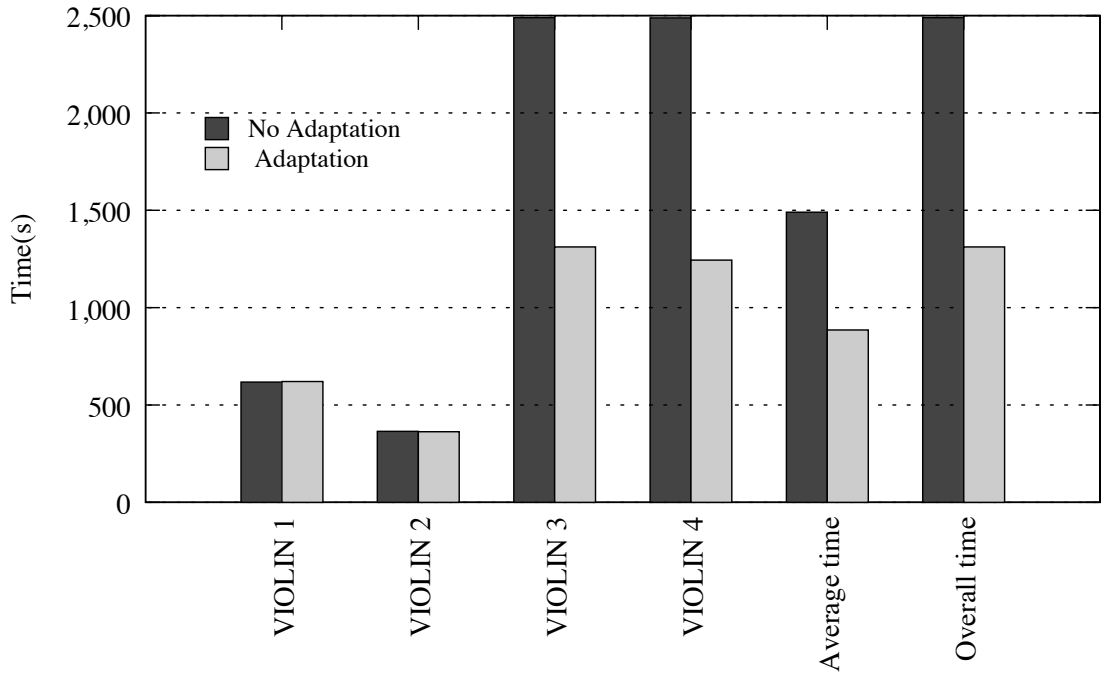


Fig. 6.7. VIOLIN environment Adaptation Scenario 2: Execution time of applications running within VIOLIN environments with and without adaptation enabled.

6.5 Conclusion

We have presented the design and implementation of VIOLIN autonomic virtual computation environments for multi-domain shared infrastructures. Using VIOLIN environments, independently administered virtual computation domains flow through the massive amount of computation resources available through multi-domain shared infrastructures. We have shown the design and implementation of VIOLIN environments that allows virtual environments to adapt to the needs of their applications including the use of wide-area migration of live virtual environments. Our experiments with our nanoHUB deployment of virtual computation environments has shown significant performance and efficiency increases. With continued advancement of machine and network virtualization, as well as resource allocation policies, VIOLIN virtual computation environments will continue to increase the potential of multi-domain shared infrastructures.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this dissertation, we have presented an integrated, virtualization-based framework for federating massive amounts of cyberinfrastructure resources. By creating a layer of indirection between physical resources and software systems, virtualization provides unique opportunities to address challenging problems in cyberinfrastructure and high-throughput computing. More specifically, the VIOLIN middleware has demonstrated the power of virtualization for the creation of efficient, secure environments for utilizing cyberinfrastructure. Using VIOLIN virtual environments (Chapter 4), we are able to provide each user with the illusion of a isolated private LAN which is actually deployed across a wide-area shared cyberinfrastructure. Further, we utilize VIOLIN environments to create VioCluster which enables inter-domain cluster sharing (Chapter 5) by dynamically expanding the domain boundaries of cluster computers to utilize the resources of cooperating clusters. Finally, we use our experience gained through the creation of VioCluster and the wide-area adaptation mechanisms of VIOLIN to create fully autonomic virtual environments for shared cyberinfrastructure (Chapter 6).

Based on the integrated VIOLIN framework, we have deployed a production system on the nanoHUB infrastructure, as well as, created experimental systems used to evaluate the effectiveness of adaptation mechanisms and policies on increasing the efficiency and throughput of shared computational resources. Based on the observation and insights obtained from this platform, we have gained unique advantages in designing and evaluating advanced virtualization-based middleware for federating cyberinfrastructure resources. Currently, VIOLIN environments are a fundamental part of the production nanoHUB execution environment, and in the near future advanced

adaptation mechanisms and policies will be extended to the nanoHUB increasing its performance and benefiting all of its users.

7.2 Future Work

The integrated middleware for autonomic adaptive virtual environments presented in this dissertation has laid a solid foundation for future work. In the following we propose topics for future research:

- **Advanced resource demand prediction methods**. The key to effective resource allocation is knowing the proper amount of resources a virtual machine or environment will need in the future. In this dissertation, we have discussed two techniques for predicting future demand: (1) application-aware prediction, and (2) application-independent prediction. VioCluster (Chapter 5), uses application-aware prediction by utilizes notoriously inaccurate and often unavailable information gathered from cluster schedulers. The fully autonomic environments discussed in Chapter 6 use application-independent prediction methods by applying relatively simple time series analysis.

  In future work, we plan on developing more advanced prediction methods based on techniques found in the field of Artificial Intelligence (AI). Reinforcement Learning (RL) is an AI technique that aims to learn about an algorithm using a *black-box* approach. Reinforcement Learning techniques manipulate the values used as input to a function and record the associated output. Over time, the system can learn how the inputs effect the output without knowing the internals functionality. If we view a virtual machine as a black-box, the resource allocation as inputs, and the completed computation as output, we can use RL to manipulate the resource allocation in order to effect the virtual machine's ability to run the application.

- **Advanced Cluster throughput mechanisms**. Computational clusters have many users. Some users are skilled at coding and running efficient applications,

while others are not. In large supercomputing centers inefficient applications have actual costs seen in the need to purchase more machines and in long queue wait times endured by users. Work has been done to identify inefficient applications and notify users in the attempt to modify their behavior [100]. We intend to utilize the job efficiency information to adapt virtual environments allocation. By applying the techniques we have developed through the creation of the VIOLIN middleware, we will continue to increase the performance and efficiency of distributed shared computational platforms.

LIST OF REFERENCES

LIST OF REFERENCES

[1] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, 2001.

[2] TeraGrid, "`http://www.teragrid.org`.".

[3] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating System Support for Planetary-Scale Network Services," in *Proceedings of the Symposium on Networked Systems Design and Implementation*, March 2004.

[4] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet ," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 59–64, 2003.

[5] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences Building PlanetLab," in *Proceedings of the 7th USENIX Symposium on Operating System Design and Implementation*, November 2006.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp. 164–177, 2003.

[7] J. Dike, "User-Mode Port of the Linux Kernel," in *Proceedings of the USENIX Annual Linux Showcases and Conference*, 2000.

[8] R. Figueiredo, P. A. Dinda, and J. Fortes, "Guest Editors' Introduction: Resource Virtualization Renaissance," *IEEE Computer*, vol. 38, no. 5, pp. 28–31, 2005.

[9] VMware, "`http://www.vmware.com`."

[10] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, "From Virtualized Resources to Virtual Computing Grids: The In-VIGO System," *Future Generation Computer Systems Special section: Complex Problem-Solving Environments for Grid Computing*, vol. 21, pp. 896–909, June 2005.

[11] R. Figueiredo, P. Dinda, and J. Fortes, "A Case for Grid Computing on Virtual Machines," in *Proceedings of International Conference on Distributed Computing Systems*, May 2003.

[12] I. T. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayer, and X. Zhang, "Virtual Clusters for Grid Communities," in *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pp. 513–520, 2006.

[13] K. Keahey, K. Doering, and I. T. Foster, "From Sandbox to Playground: Dynamic Virtual Environments in the Grid," in *Proceedings of the 5th International Workshop in Grid Computing*, pp. 34–42, November 2004.

[14] A. Shoykhet, J. Lange, and P. Dinda, "Virtuoso: A System For Virtual Machine Marketplaces," Tech. Rep. NWU-CS-04-39, Northwestern Univeristy, July 2004.

[15] X. Jiang and D. Xu, "VIOLIN: Virtual Internetworking on OverLay INfrastructure," Tech. Rep. TR 03-027, Purdue University Department of Computer Science, July 2003. in LNCS Vol. 3358, Springer.

[16] M. Kallahalla, M. Uysal, R. Swaminathan, D. E. Lowell, M. Wray, T. Christian, N. Edwards, C. I. Dalton, and F. Gittler, "SoftUDC: A Software-Based Data Center for Utility Computing," *IEEE Computer*, vol. 37, no. 11, pp. 38–46, 2004.

[17] A. I. Sundararaj and P. A. Dinda, "Towards Virtual Networks for Virtual Machine Grid Computing," in *Proceedings of USENIX Virtual Machine Research and Technology Symposium*, pp. 177–190, 2004.

[18] A. Sundararaj, A. Gupta, and P. Dinda, "Increasing Application Performance In Virtual Environments Through Run-time Inference and Adaptation," in *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*, July 2005.

[19] P. Ruth, X. Jiang, D. Xu, and S. Goasguen, "Virtual Distributed Environments in a Shared Infrastructure," *IEEE Computer*, vol. 38, pp. 63–69, May 2005.

[20] X. Jiang and D. Xu, "SODA: A Service-On-Demand Architecture for Application Service Hosting Utility Platforms," in *Proceedings of The 12th IEEE International Symposium on High Performance Distributed Computing*, June 2003.

[21] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle, "Dynamic Virtual Clusters in a Grid Site Manager," in *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, p. 90, 2003.

[22] H. Liu and M. Parashar, "Enabling Self-Management of Component Based High-Performance Scientific Applications," in *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*, 2005.

[23] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen, "Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure," in *Proceedings of the 3rd IEEE International Conference on Autonomic Computing*, June 2006.

[24] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation*, May 2005.

[25] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast Transparent Migration for Virtual Machines," in *USENIX Annual Technical Conference, General Track*, pp. 391–394, April 2005.

[26] "`http://www.nanohub.org`."

[27] J. Fortes, R. Figueiredo, and M. Lundstrom, "Virtual Computing Infrastructures for Nanoelectronics Simulation," in *IEEE*, vol. 93(10), pp. 1839–1847, 8 2005.

[28] P. Ruth, P. McGachey, and D. Xu, "VioCluster: Virtualization for Dynamic Computational Domains," in *Proceedings of the IEEE International Conference on Cluster Computing*, September 2005.

[29] D. E. Atkins, K. K. Droegemeier, S. I. Feldman, H. Garcia-Molina, M. L. Klein, D. G. Messerschmitt, P. Messina, J. P. Ostriker, and M. H. Wright, "Revolutionizing Science and Engineering Through Cyberinfrastructure," 2003.

[30] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency and Computation: Practice & Experience*, vol. 17, pp. 323–356, February 2005.

[31] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard," Tech. Rep. UT-CS-94-230, The University of Tennessee and Oak Ridge National Laboratory, 1994.

[32] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 2, pp. 115–128, 1997.

[33] M. Varian, "VM and the VM Community, Past, Present, and Future," in *SHARE 89 Sessions 9059-9061*, 1997.

[34] F. J. Corbat;, M. Merwin-Daggett, and R. C. Daley, "An Experimental Time-Sharing System," *Classic Operating Systems: From Batch Processing to Distributed Systems*, pp. 117–137, 2000.

[35] R. J. Creasy, "The Origin of the VM/370 Time-Sharing System," *IBM Journal of Research and Development*, vol. 25, no. 5, pp. 483–490, 1981.

[36] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig, "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization," *Intel Technology Journal*, 2006.

[37] Advanced Micro Devices, "`http://www.amd.com/us-en/Processors/ProductInformation/`.".

[38] J. Casazza, M. Greenfield, and K. Shi, "Redefining Server Performance Characterization for Virtualization Benchmarking," *Intel Technology Journal*, 2006.

[39] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," in *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pp. 1–14, June 2001.

[40] C. Waldspurger, "Memory Resource Management in VMware ESX Server," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, December 2002.

[41] Z. Amsden, D. Arai, D. Hecht, A. Holler, and P. Subrahmanyam, "VMI: An Interface for Paravirtualization," in *Proceedings of the Ottawa Linux Symposium*, July 2006.

[42] I. Ahmad, J. Anderson, A. Holler, R. Kambo, and V. Makhija, "An Analysis of Disk Performance in VMware ESX Server Virtual Machines," in *Proceedings of IEEE International Workshop on Workload Characterization, WWC-6*, pp. 65–76, October 2003.

[43] XenSource, Inc., "A Performance Comparison of Commercial Hypervisors." `http://www.xensource.com/files/hypervisor\_performance\_comparison\_1\_0\_5\_with\_esx-data.pdf`.

[44] Parallels Inc., "`http://www.parallels.com`.".

[45] Virtual PC, "`http://www.microsoft.com/windows/products/winfamily/virtualpc/default.mspx`.".

[46] Qumranet Inc. White Paper, "KVM: Kernel-based Virtualization Driver." 2006.

[47] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," pp. 41–46.

[48] M. Larabel, "Linux KVM Virtualization Performance," *Phoronix*, January 2007.

[49] VServer Community, "`http://linux-vserver.org/paper`.".

[50] SWSOFT White Paper, "An Introduction to OS Virtualization and Virtuozzo." `http://www.swsoft.com/r/pdfs/vz/whitepapers/VZ-Overview.pdf`.

[51] OpenVZ, "`http://openvz.org`.".

[52] SWSOFT White Paper, "Live Migration." `http://www.swsoft.com/r/pdfs/vz/whitepapers/Virtuozzo\_Live\_Migration.pdf`.

[53] P. Kamp and R. Watson, "Jails: Confining the Omnipotent Root." `http://docs.freebsd.org/44doc/papers/jail/jail.html`.

[54] D. Price and A. Tucker, "Solaris Zones: Operating System Support for Consolidating Commercial Workloads," in *Proceedings of the 18th USENIX Conference on System Administration*, pp. 241–254, 2004.

[55] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J. N. Matthews, "Xen and the Art of Repeated Research," in *Proceeding of USENIX Annual Technical Conference, FREENIX Track*, pp. 135–144, 2004.

[56] K. Adams and O. Agesen, "A Comparison of Software and Hardware Techniques for x86 Virtualization," in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pp. 2–13, 2006.

[57] A. Sinha and A. P. Chandrakasan, "Energy Efficient Real-time Scheduling," in *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, pp. 458–463, 2001.

[58] J. Lange, A. Sundararaj, and P. Dinda, "Automatic Dynamic Run-time Optical Network Reservations," in *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*, 2005.

[59] B. Lin and P. Dinda, "VSched: Mixing Batch and Interactive Virtual Machines Using Periodic Real-time Scheduling," in *Proceedings of ACM/IEEE SC 2005 (Supercomputing)*, 2005.

[60] A. Ganguly, A. Aagrawal, P. O. Boykin, and R. Figueiredo, "IP over P2P: Enabling Self-configuring Virtual IP Networks for Grid Computing," in *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium*, 2006.

[61] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "WOW: Self-Organizing Wide Area Overlay Networks of Virtual Workstations," in *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, June 2006.

[62] D. Wolinsky, A. Agrawal, P. O. Boykin, J. Davis, A. Ganguly, V. Paramygin, P. Sheng, and R. Figueiredo, "On the Design of Virtual Machine Sandboxes for Distributed Computing in Wide Area Overlays of Virtual Workstations," in *First Workshop on Virtualization Technologies in Distributed Computing (VTDC), with Supercomputing (SC07)*, 2007.

[63] L. Zhang and D. Ardagna, "SLA Based Profit Optimization in Autonomic Computing Systems," in *Proceedings of the 2nd international conference on Service oriented computing*, pp. 173–182, 2004.

[64] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, vol. 36, pp. 255–270, December 2002.

[65] M. Ripeanu, M. Bowman, J. S. Chase, I. Foster, and M. Milenkovic, "Globus and PlanetLab Resource Management Solutions Compared," in *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, pp. 246–255, 2004.

[66] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. Vahdat, "Model-Based Resource Provisioning in a Web Service Utility," in *Proceedings of USENIX Symposium on Internet Technologies and Systems*, March 2003.

[67] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, "Virtual Machine Hosting for Networked Clusters: Building the Foundations for "Autonomic" Orchestration," in *Proceedings of the First International Workshop on Virtualization Technology in Distributed Computing*, November 2006.

[68] L. Grit, J. Chase, D. Irwin, and A. Yumerefendi, "Shirako: Virtual Machine Hosting for Federated Clusters," in *Seventh USENIX Symposium on Operating Systems Design and Implementation*, November 2006.

[69] L. Ramakrishnan, L. Grit, A. Iamnitchi, D. I. A. Yumerefendi, and J. Chase, "Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control," in *Proceedings of the 19th Annual Supercomputing Conference*, November 2006.

[70] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. Yocum, "Sharing Networked Resources with Brokered Leases," in *Proceedings of USENIX Technical Conference*, May 2006.

[71] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing Risk and Reward in a Market-Based Task Service," in *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, pp. 160–169, June 2004.

[72] K. Keahey, I. T. Foster, T. Freeman, X. Zhang, and D. Galron, "Virtual Workspaces in the Grid," in *Proceedings of Euro-Par*, pp. 421–431, 2005.

[73] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid," *Scientific Programming Journal*, vol. 13, no. 4, 2005, Special Issue: Dynamic Grids and Worldwide Computing, pp. 265–275, 2005.

[74] B. Sotomayor, K. Keahey, and I. Foster, "Overhead Matters: A Model for Virtual Resource Management," in *Proceedings of the First International Workshop on Virtualization Technology in Distributed Computing*, 2006.

[75] Beowulf, "`http://www.beowulf.org`."

[76] Portable Batch Scheduler, "`http://www.openpbs.org`."

[77] Torque, "`http://www.clusterresources.com/pages/products/torque-resource-manager.php`."

[78] Seti@home, "`http://setiweb.ssl.berkeley.edu/`."

[79] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," in *Open Grid Service Infrastructure WG*, Global Grid Forum, June 2002.

[80] P. T. Bulhes, C. Byun, R. Castrapel, and O. Hassaine, "N1 Grid Engine 6 Features and Capabilities." Sun Microsystems White Paper, `http://www.sun.com/products-n-solutions/edu/whitepapers/pdf/N1GridEngine6.pdf`.

[81] S. Adabala, A. M. Matsunaga, M. O. Tsugawa, R. J. O. Figueiredo, and J. A. B. Fortes, "Single Sign-On in In-VIGO: Role-Based Access via Delegation Mechanisms Using Short-Lived User Identities," in *Proceedings of 18th International Parallel and Distributed Processing Symposium*, April 2004.

[82] V. Sanjeepan, A. Matsunaga, L. Zhu, H. Lam, and J. A. Fortes, "Service-oriented, scalable approach to grid-enabling of legacy scientific applications," in *Proceedings of 2005 International Conference on Web Services*, pp. 553–560, July 2005.

[83] A. Ganguly, D. Wolinsky, P. O. Boykin, and R. Figueiredo, "Decentralized Dynamic Host Configuration in Wide-Area Overlay Networks of Virtual Workstations," in *Workshop on Large-Scale and Volatile Desktop Grids*, March 2007.

[84] A. Matsunaga, M. Tsugawa, S. Adabala, R. Figueiredo, H. Lam, and J. Fortes, "Science gateways made easy: the in-vigo approach," in *Workshop on Science Gateways, Global Grid Forum*, 2005.

[85] M. Zhao, J. Xu, and R. Figueiredo, "Towards Autonomic Grid Data Management with Virtualized Distributed File Systems," in *International Conference on Autonomic Computing*, 2006.

[86] M. Zhao, J. Zhang, and R. Figueiredo, "Distributed File System Support for Virtual Machines in Grid Computing," in *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, June 2004.

[87] M. Zhao, J. Zhang, and R. Figueiredo, "Distributed File System Virtualization Techniques Supporting On-Demand Virtual Machine Environments for Grid Computing," *Proceedings of Cluster Computing*, vol. 9, pp. 45–56, January 2006.

[88] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo, "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing," in *the Proceedings of the ACM/IEEE Super Computing Conference*, p. 7, 2004.

[89] D. Hepema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne, "A Worldwide Flock of Condors: Load Sharing Among Workstation Clusters," *Future Generation Computer Systems*, vol. 12, no. 1, Special issue: resource management in distributed systems, pp. 53–65, 1996.

[90] D. P. Anderson, "Public Computing: Reconnecting People to Science," in *Proceedings of Conference on Shared Knowledge and the Web*, November 2003.

[91] I. Foster and C. Kesselmann, "Globus: A Toolkit-based Grid Architecture," *The Grid: Blueprints for a New Computing Infrastructure*, pp. 259–278, 1999.

[92] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, "An Updated Set of Basic Linear Algebra Subprograms (BLAS)," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 135–151, 2002.

[93] Message Passing Interface Forum, "MPI-2 Journal of Development." `http://www.mpi-forum.org/docs/mpi-20-jod.ps`, 1997.

[94] J. Dongarra, J. Bunch, C. Moler, and G. Stewart, "LINPACK Users Guide." SIAM, 1979.

[95] G. Klimeck, F. Oyafuso, T. B. Boykin, R. C. Bowen, and P. von Allmen, "Development of a Nanoelectronic 3-D (NEMO 3-D) Simulator for Multimillion Atom Simulations and Its Application to Alloyed Quantum Dots," *Computer Modeling in Engineering and Science*, vol. 3, no. 5, pp. 601–642, 2002.

[96] The Hebrew University – Parallel Systems Lab, "`http://www.cs.huji.ac.il/labs/parallel/workload/`."

[97] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart, "An Architectural Approach to Autonomic Computing," in *Proceedings of the IEEE International Conference on Autonomic Computing*, 2004.

[98] G. Tesauroa, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White., "A Multi-Agent Systems Approach to Autonomic Computing," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 464–471, 2004.

[99] G. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*. Oakland, CA: Holden-Day, second edition ed., 1976.

[100] Mississippi Center for Supercomputing Research, "`http://www.mcsr.olemiss.edu/allMCSRCrntJb.php`.".

VITA

VITA

Paul Michael Ruth was born in Urbana, Illinois and spent most of his childhood in Escondido, California. He attended San Pasqual High School where he way always interested in mathematics and science but spent most of his time playing soccer. He graduated from high school in 1994 and went on to attend Baker University in Baldwin City, Kansas. This unlikely choice of college was due mostly to his desire to experience life on his own and the opportunity to play soccer on an athletic scholarship. While a undergraduate, he had various jobs within the university including lab assistant and working for computer services, administering, configuring, and installing computers and computer networks. In 1998 he graduated with a Bachelor of Science degree in mathematics and computer science. After graduating from Baker University, Paul spent one year working as a contractor with Sprint in Kansas City. In 1999, he arrived at Purdue University where he spent many semesters as a teaching and research assistant. Much of his work as a research assistant was in developing and deploying the nanoHUB gateway. As a result of his work on the nanoHUB he is a recipient of the 2007 Halstead Award for his contributions to virtualization middleware for distributed and parallel computing. As a research assistant under Dongyan Xu, he earned his Doctor of Philosophy in computer science in August of 2007. In the fall of 2007 he joined the faculty of the Department of Computer and Information Science at the University of Mississippi in Oxford, Mississippi as an assistant professor.