Xen and the Art of Distributed Virtual Machine Management

Undergraduate Honors Thesis

By Adam Zacharski
Department of Computer Sciences
The University of Texas at Austin
adam@zacharski.org

Supervising Professor: Dr. Greg Lavender

Table of Contents

List of Figures

Abstract

In this thesis report, a method for automating the creation and management of many virtual machines across multiple physical machines is described. By using the Xen paravirtualization software as a base, this method adds tools to create new virtual machine images and allows for live migration without any human intervention. This method uses management agents that run on the physical and virtual machines. There is also a management GUI that can be run from any computer on the network. All of these agents (the management clients and the GUI) communicate using a publish/subscribe based messaging system. For example, when a machine is overloaded the server agent on that machine broadcasts a message to the publish/subscribe message bus that contains a request for help. This system was evaluated on a network of four dual processor dual core servers. Multiple high load virtual machines were started on a single physical machine. The system was then monitored to make sure that the virtual machines were evenly distributed among the other physical machines on the network. The test results show that this method for virtual machine management could be used for the management of a few hundred virtual machines.

1. Introduction

When computers first came on the scene, they were so expensive and bulky that many people had to share a single computer. In the past few years, computers have become more commonplace and it is not uncommon to have more than one. The raw computing power that today's multi-core CPUs provide is very rarely needed. For example, Microsoft Office Word 2007 requires a 500MHz CPU and Internet Explorer 7 requires a 233 MHz CPU. While these numbers might be a little too hopeful, gigahertz machines have been around for over 5 years. An average computer user is using less than an eighth of the available computing power of a typical desktop. This underutilized resource is one motivation to have many virtual machines on a single physical machine.

With only a few virtual machines per physical machine it is fairly easy to manage them all, but with the introduction of chips with 80 cores like the one just demonstrated by Intel[1], managing all of these virtual machines quickly becomes impractical. Another problem appears with the introduction of racks or clusters of physical machines. Some of the physical machines could have a full load while others could have no load at all. Live migration of virtual machines (migration without interruption of service) has been around for a while but it requires human intervention. With the ability to migrate virtual machines around, another possibility arises, such as live migration of virtual machines off of a physical machine to do software upgrades or hardware maintenance and then moving the virtual machines back without users of the virtual machines being able to tell the difference.

The idea of virtual machines would be useful in university instructional and research environments, especially in computer science. For example, consider an operating systems class. Without virtualization, if a student was required to write an operating system or modify the kernel, that student would need a dedicated computer or a software simulator for a physical machine. Even with that dedicated resource, work on these projects would be difficult since programming errors require multiple slow restarts or at least reimaging of the test machine. With virtualization both of these problems are solved. Xen, a software paravirtualization layer that allows multiple virtual machines to be run on a single physical machine, allows multiple students to be working on a single physical machine by providing each student with two virtual machines, one virtual machine with Linux for development and another for testing their new OS. Moreover, a crash of any student's virtual machine does not affect any other student's virtual machine or the student's development machine. For debugging, a Xen extension could be easily written using the debugging extension for UML[2], provided by the University of Michigan, as a guide. This extension would provide for easy debugging of virtual machines by using a GDB like interface with rollback capability. Another use of virtualization in the computer science curriculum would be in the areas of web development and computer networking. Not only could each student have their own server, but they could test distributed web applications by having multiple virtual machines and virtual network interfaces. In the more general case of computer labs, it would allow for more flexibility. For example, a student would have their own custom computing environment that would automatically be migrated to any physical machine at which a student chooses to work. By using Xen instead of sharing home directories with NFS a user would be able to choose which operating system and applications they wanted. Providing students with the ability to experiment and modify their computing environment enables students to develop practical skills that complement theoretical coursework.

The purpose of this project is to develop a system that is able to greatly simplify the management of these virtual machines by creating:

1. A central interface for the creation and control of the virtual machines
2. An automated system that will keep virtual machines as evenly distributed as possible (allowing overloaded machines on the network to automatically offload virtual machines to underutilized physical machines).

The system consists of three agents (server agent, virtual agent and configuration agent) and a management GUI. By using a publish/subscribe message system all of these agents are able to multicast messages to each other. The publish/subscribe system that was used in this system was Sun's Java Message Service (JMS)[3]. This message service consists of a message queuing server running on a central machine that forwards messages from the publishers to the subscribers.

We evaluated the system by running it on a rack of servers running 6.06 Ubuntu Linux with a Sun Solaris 10 x64 machine as the JMQ server. The physical machines were made up of two x2100s (a 2.2GHz dual core Operon processor, 4gb of ram and two 250gb disks) and four v20zs (two 2.2GHz dual core Operon processors, 16gb of ram and two 300gb disks). All of the virtual machines used for testing purposes were also running Ubuntu. While the system was tested on Ubuntu, this is a general-purpose solution that will work with any platform that is compatible with Xen 3.x.x and has a Java Runtime Environment (JRE) of at least version 1.5.

2. Virtualization

Virtualization is the abstraction of the physical resources on a computer. With this layer of abstraction, it becomes possible to simulate multiple copies of devices like CPUs, memory and even disks. This abstraction layer can provide a simplified view of the hardware as well as allowing for strict access policies. There are many reasons why virtualization is a good thing. If your virtualization software is well written, any buggy guest operating system will not be able to crash the entire machine due to the sandbox that the guest operating system is being kept in (isolated from all other virtual machines). Virtualization allows for increased reliability and uptime. For example, if you are running a web server on a physical server and you need to upgrade the hardware, the services would have some downtime. However, if the server was running in a virtual machine it would be possible to migrate it to another physical machine and therefore services are not visibly disrupted during hardware upgrades. If multiple virtual machines are allowed to run on a single physical machine it allows for better utilization of the hardware. This is already seen in industry with virtual hosting services. If you buy web hosting, chances are that you are sharing the physical machine with many other people. Lastly, by definition, virtualization provides an abstracted view of the hardware on the physical machine which allows for a simplified driver set on the virtual machines. There are several alternative methods that allow virtualization to be accomplished: software virtualization, paravirtualization and hardware virtualization support.

Software virtualization requires the virtualization software to handle requests that are restricted to the host OS, all other tasks like program execution run with no intervention. This is opposed to emulation which intercepts every instruction and translates it to an appropriate instruction for the architecture; an example of this is PearPC which is an architecture-independent emulator for the PowerPC platform[4]. One of the most common software virtualization products is VMware[5]. The VMware architecture is shown below in Figure 1.

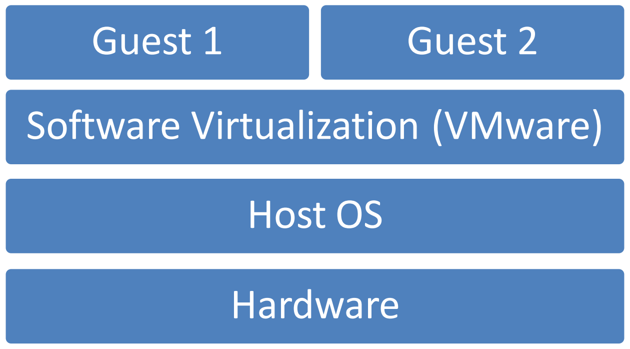


Figure 1: Software Virtualization

On modern processors, a software virtualization technique called trap-and-emulate can be used. When the guest operating system attempts to perform an instruction that is restricted a trap is generated. The virtualization layer is then called by the operating system with information about what the guest was trying to do. The virtualization layer then performs the necessary operations to make it appear to the guest operating system as if its instruction had succeeded. Since the x86 architecture does not support the typical trap-and-emulate method, VMware uses binary translation. In VMware the guest operating system runs in user space and has its code translated on the fly so that it no longer contains privileged instructions.

In paravirtualization, a sub-type of software virtualization, the virtualization layer runs right on top of the hardware. The host operating system, Dom0, runs on top of this layer and contains all of the management facilities for the virtual machines. As opposed to software virtualization, a guest operating system, running as DomU, is aware that it is being run in a virtual machine and so it behaves somewhat differently. For example, a DomU that is aware that it is a virtual machine uses hypercalls and events instead of trying to do a system call and trapping. In addition, paravirtualization allows for specialized methods of device access, like Xen IO rings. This is further described in section 3 below.

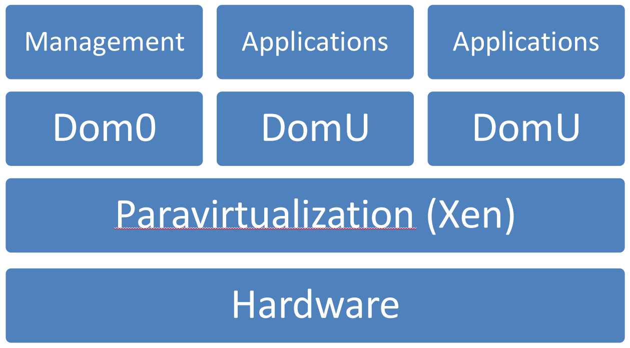


Figure 2: Paravirtualization

Modern processors come with hardware virtualization support. This includes Intel's VT^{TM} technology and AMD-V^{TM} ("Pacifica") technology. Hardware support for virtualization greatly increases the speed of virtualization as well as security. AMD's IOMMU technology is a replacement for the typical MMU [6]. This allows for guest operating systems to directly access hardware and memory while the IOMMU guarantees that they access proper memory locations and not memory locations of other domains. IOMMU can aid in memory management by allowing for non-contiguous memory locations.

Paravirtualization like Xen can take direct advantage of hardware virtualization since Xen runs directly on top of the hardware.

3. Xen

3.1 Overview

Xen was created by Ian Pratt at the University of Cambridge[7]. The main difference between VMware and Xen is that Xen makes use of a hypervisor. In this report we are using hypervisor to refer to a software layer that is running directly on the hardware with the host and guest operating systems running on top of this layer. The guest operating system uses a special API for calls that would normally be called in ring 0.

3.2 Xen IO

Instead of emulating IO devices on common interfaces like SCSI, Xen provides the notion of IO rings. The virtual domain places a request on the ring which contains an IO descriptor. The IO descriptor contains information for the request including the location IO data buffer that was allocated by the virtual domain. Xen advances around the ring handling requests. Two shared pointers and two private pointers are used to keep track of the sections of the ring: request queue, outstanding descriptors, response queue and unused descriptor.[7]

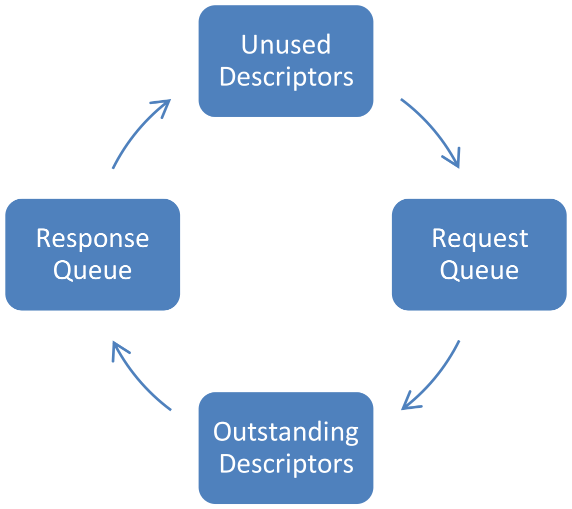


Figure 3: Xen IO Ring

3.3 Migration

One of the most beneficial features of Xen is that it allows for the live migration of virtual machines between physical machines in a local area network. To make this possible Xen has to transfer memory contents and information about the domain's configuration. The disk images of the currently running domain also have to be accessible on both machines. To transfer memory from one machine to the other there are three stages. The first stage is known as the push phase. This phase pushes all of the memory that is not being updated frequently across the network. If one of the pages that is transferred is changed, it will be marked and transferred again later. The next stage is the "Stop-and-Copy" phase. During this phase the virtual machine is paused and a small amount of memory that is being frequently updated is transferred. After this stage is completed the virtual machine will be resumed on the new host. The last stage in the memory migration is the "Pull" phase. During this phase the rest of the memory is copied and if the virtual machine page faults that page is transferred. When all memory transfer is complete the source machine removes the state of the transferred machine.

In order for the network traffic to be rerouted to the new physical machine there are two options. The first method sends an unsolicited ARP reply from the virtual domain with the new location. Because this type of ARP broadcast is sometimes used for IP spoofing some routers do not allow it. If the operating system is aware of its migration then it can send directed ARP to specified hosts. The other method is possible if the virtual machine is on switched network because after a short period of time the switch will recognize that the virtual machine has moved and will remap its MAC address.[8]

The whole migration time varies on the activity of the virtual domain that is being migrated. If the virtual machine is under heavy load, SPECWeb99, migration can take up 71 seconds with a downtime of 210ms. Otherwise, if the load on the server is light, Quake 3 server, migration will take a little less than 7 seconds with a downtime of 70ms.[1] As a worst-case scenario, a domain that is dirtying pages faster than they can be transferred over the network will be suspended. This will result in a 30-second migration time but a 3.5-second downtime.[8]

The architecture of Xen is shown in Figure 2 above. The main difference between Xen's architecture and that of VMware is that Xen takes the place of the host operating system right on top of the hardware layer while VMware is running on top of the host OS. Xen provides tools to monitor the virtual machines that are currently running, one of which is xmtop. Xmtop provides a view of the CPU usage of all of the virtual machines and which physical CPU they match to. It also provides a look at the network traffic that is going to and from the virtual machines. Lastly it provides the status of the virtual machines, whether they're paused, running, blocked, etc.

3.4 Xen Configuration

Creating a Xen configuration file is straight forward. For our purposes we only use six parameters.

```
1:      kernel = "/boot/vmlinuz-2.6-xen"
2:      disk = ['phy:/dev/gnbd/small-1-disk,sda1,w','phy:/dev/gnbd/small-1-swap,sda2,w' ]
3:      memory = 512
4:      name = small-1'
5:      vif = [ 'bridge=xenbr0' ]
6:      root = "/dev/sda1 ro"
```

Line 1 specifies which Linux kernel we want the virtual machine to be loaded with.

Next comes the disks that we want to have accessible to the virtual machine. These disks can either be real partitions or flat files that are residing in the file system of the host domain. In our case we used partitions that were controlled by the Global Network Block Device (GNBD) file system software.[9]

Line 3's parameter specifies how much memory should be allocated to a virtual machine when it is started.

All line 4 does is specify the name that should appear in the virtual domain lists when referring to the machine started with this configuration file.

Line 5 specifies how we want the networking to be configured. In this case we are bridging the virtual adaptor with bridge xenbr0.

[1] Both SPECweb99 and Quake 3 are commonly used to benchmark systems.[14]

The last parameter is similar to the root parameter in boot loaders that lets the kernel know where it should look for needed data on its virtual disks.

4. System Setup

4.1 Servers

For our testing purposes we had six x64 Opteron machines. Two of the machines were Sun SunFire x2100s and the remaining four were Sun SunFire v20z’s. One of the x2100s was used as a Java Message Server (JMS) control system server and other was used to host the configuration files and the disk images for the virtual machines by acting as a network attached file system. It would be possible to make the JMS and the configuration server the same machine because neither task is overly strenuous but we wanted to separate the management machine from the file server. The four v20z's were used for running the virtual machines because they each have 4 processors (2 dual core Opteron 275s) with 16 gigabytes of ECC memory.

4.2 Networking

In our experiments, we found that it was beneficial to have two separate networks to isolate network traffic.

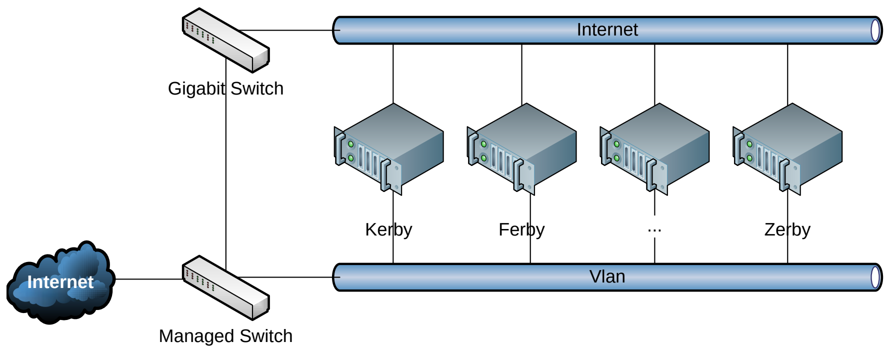


Figure 4: Network Layout

Figure 4 shows how the machines in our lab are networked. Every server has 4 gigabit Ethernet ports, two of which are used to connect to both a managed switch and a gigabit switch. The gigabit switch connects all of the machines to the Internet, while the managed switch connects all of the machines onto the Xen VLAN. For our purposes we found that it is beneficial to filter the Internet through the managed switch and only allow traffic from specified MAC addresses through. The main reason is that while trying to get virtual machine bridging sorted out, our campus network would detect an unauthorized MAC address (a virtual machine) and disable our network port.

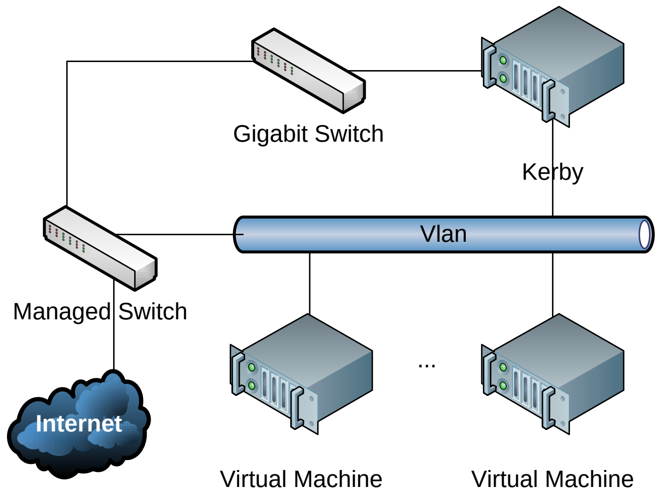


Figure 5: VLAN Layout

The virtual machines are connected to the Xen VLAN by bridging their network interface to the physical machine's network device. When a virtual machine is first started it requests an IP address from a DHCP server running in the managed switch. The managed switch provides an ip address on the 10.0.0.* subnet, with the default gateway being Kerby. In order for the virtual machines to access the internet, NAT is used. This allows for the traffic to be easily monitored and does not require that each of the virtual machines have their own public IP address. In order to access the virtual machines one needs to first connect using SSH to one of the physical machines and then to a virtual machine. This setup was chosen for two reasons. First, for testing purposes there was no reason to be using public IP addresses for all of the virtual machines. The second is that the reduced accessibility hides the virtual machines from the Internet so we are more secure from attack. In practice, one could simply use a firewall if enough public IP addresses for all virtual machines were available.

4.3 Additional Hardware

It would be beneficial to have a network attached storage device to centralize all file system management, but our current test configuration uses a small file server providing network access to the file system for each virtual machine.

5. Management of Virtual Machines

In an effort to make management as automated as possible, we attempted to automate the installation of Ubuntu onto the physical servers. This would allow for organizations using the system to plug in additional physical machines and these machines would be automatically imaged with the necessary software. Due to complicated installation requirements such as custom kernel modules and LVM partitions, Ubuntu's incomplete kickstart support was not a great solution. The next best solution is to make an image of a fresh installation and use that to image new machines. Because this is a straightforward operation, it was felt that it was unnecessary to implement it on test setup.

To make this project work the way it was intended, there are many pieces of software that had to come together. At the base of the structure are the operating systems, Ubuntu and Solaris. For the IO layer we are using both NFS and GNBD. NFS is being used for the sharing of the configuration files and the management software. GNBD is being used for the sharing of the virtual machine images. GNBD exports partitions at the block level. This turned out to be necessary because of the incompatibility of NFS and loopback devices. In order for GNBD to work across multiple machines, all of the machines need to be set up in a cluster. The cluster uses the GULM lock manager and a configuration distribution server[10]. LVM is used for the managing of the virtual machine images. LVM proved useful because it allows for dynamic resizing of partitions, dynamically adding more drives to the pool and partition fragmentation is not an issue.

The Java Message Service (JMS) provides facilities for both a publish/subscribe and a reliable queue-based message service using Java Message Queue (JMQ). For this project all of the messaging was done using the publish/subscribe approach. In the publish/subscribe model, publishers connect to the JMQ server under a certain topic. Every message that they send will be forwarded to all of the subscribers of that topic. This model was chosen because it does not matter to a publisher whether there is one subscriber or a thousand, the JMQ server takes care of it all. This allows for efficient scaling by allowing an administrator to simply plug in a new server without the need to update a server list on all of the physical machines. JMS provides what is known as durable subscribers which guarantees that the subscriber will receive the message even if they are offline when the message is sent. For our purposes this is not necessary because we don't want to build up a list of commands for a machine that is currently offline. For example you wouldn’t want a message telling the physical machine to start a virtual domain to be kept because most likely the administrator would have started that virtual domain on a different physical machine when they realized the initial physical machine was down.

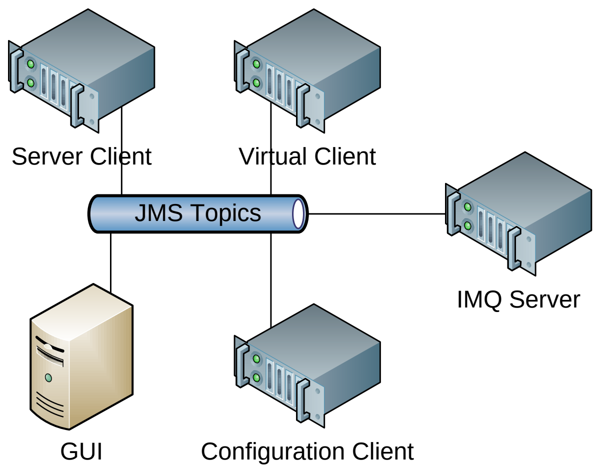


Figure 6: JMS Layout

All of the JMS messages are sent over the Xen VLAN. There is the possibility that a rogue virtual machine could forge control messages. To solve this problem one could use the facilities provided by the JMS framework to encrypt and sign messages. One could also use VLAN tagging or just simply keep the two networks separate.

6. Management GUI

The purpose of the management GUI is to provide a central location for the administrator to monitor and control the physical and virtual machines on the network. There are many composites that make up the GUI. Here I present a quick run through. For the following screenshots, the system has been setup with 5 configuration files (small-1 through small-5), three physical servers (Ferby, Gerby and Herby) and two running virtual machines (small-1 and small-2).

6.1 Configuration Creation

The configuration creation composite window shown in Figure 7, provides a simple interface to create additional virtual machines. The administrator enters the name, memory, swap and hard disk space that they want the new virtual machine to have. They then press “Create Domain” and a message is sent over the JMS publish/subscribe message bus to the configuration server containing the entered information. The left side of this window provides a quick overview of the existing configuration files, the servers on the network, domains running on the servers, and domains that are running the virtual agent. In this situation a server refers to a physical machine where the agent is running in Dom0, domains refers to all of the DomUs running on all of the servers and test domains refers to all DomUs that are running the virtual agent.

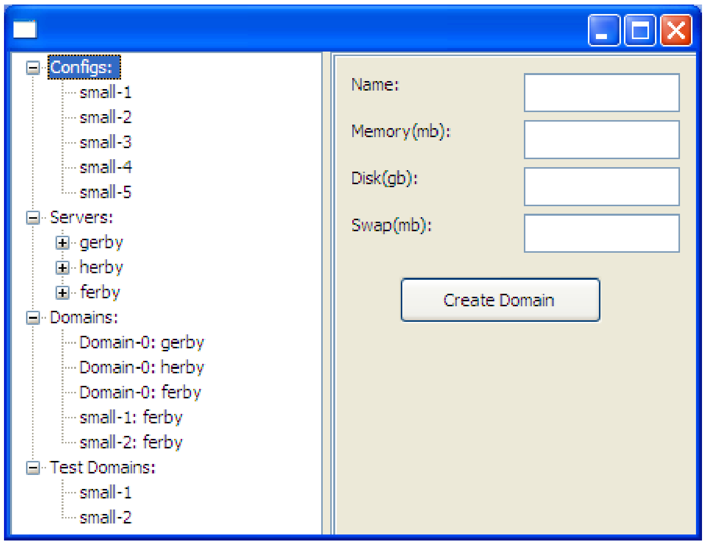
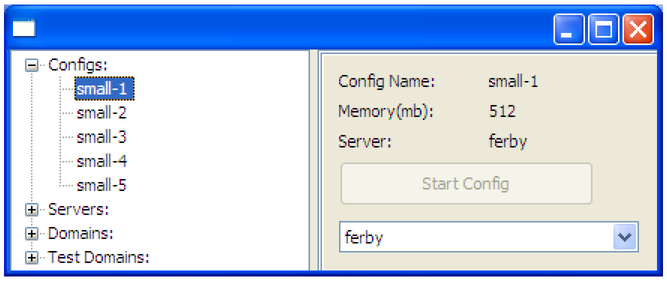


Figure 7: Configuration Composite

6.2 Configuration View

The configuration view composite (Figure 8) provides information about an existing virtual domain including its memory allocation and if it is running, which server it is running on. It gives the administrator the ability to start the virtual machine on any of the currently running servers.

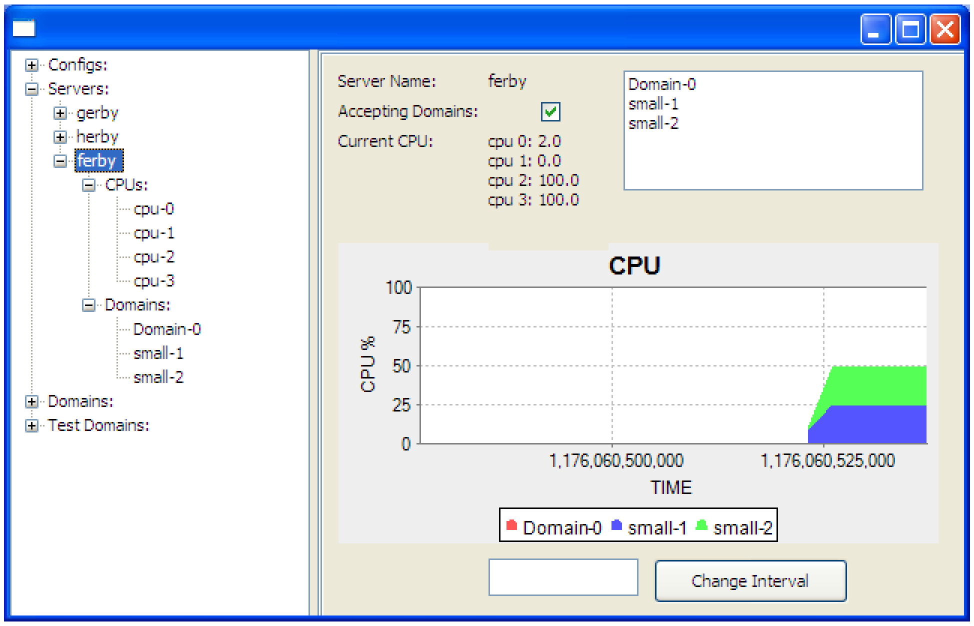


Config Name: small-1
Memory(mb): 512
Server: ferby

Figure 8: Configuration View Composite

6.3 Server Composite

The server composite gives general information about the server. It displays a list containing the currently running domains, the load of the physical CPUs and a graph showing the CPU of the various domains over time. Figure 9 shows physical machine Ferby with virtual domains small-1 and small-2 each using 100% of the core they're running on.



Server Name: ferby
Accepting Domains: ✓
Current CPU: cpu 0: 2.0
cpu 1: 0.0
cpu 2: 100.0
cpu 3: 100.0

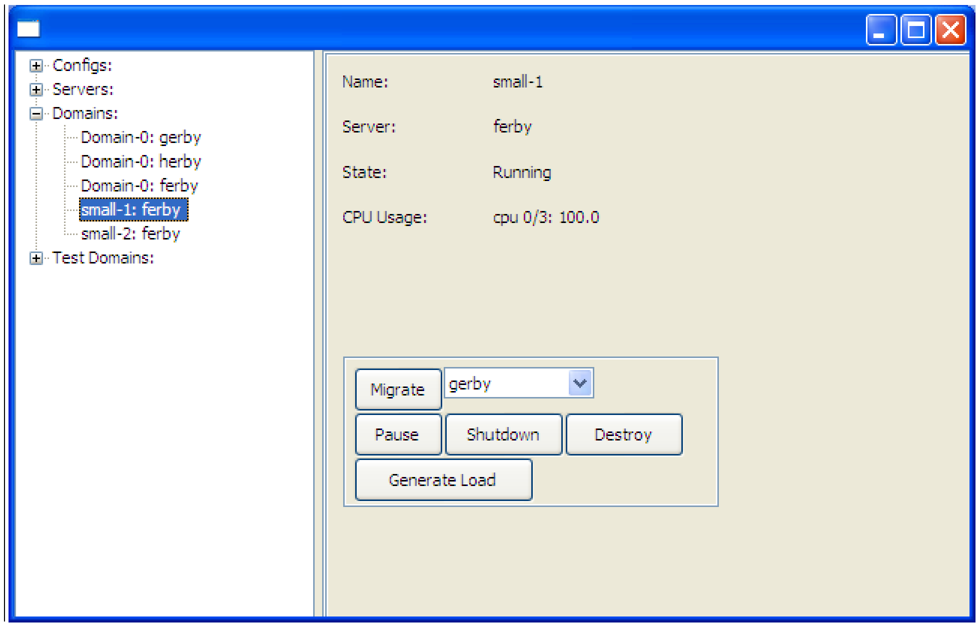
Domain-0
small-1
small-2

🟥 Domain-0 🟦 small-1 🟩 small-2

Figure 9: Server Composite

6.4 Domain Composite

The domain composite gives information about a specific domain. It shows the virtual to physical CPU mapping as well as load. It also provides the administrator the ability to migrate the domain to another server on the network. In addition, it allows the administrator to pause/unpause, shutdown and destroy the domain. Lastly, if the domain is running the virtual agent the "Generate Load" button starts a program on the domain that will drive up CPU usage to 100% for a couple of minutes. This is useful for testing purposes because we can drive up the load on a physical machine and force the system to live migrate one of the domains to another physical machine. For example, in Figure 10 you can see that small-1 is currently generating load and taking 100% of physical CPU 3.



Name: small-1

Server: ferby

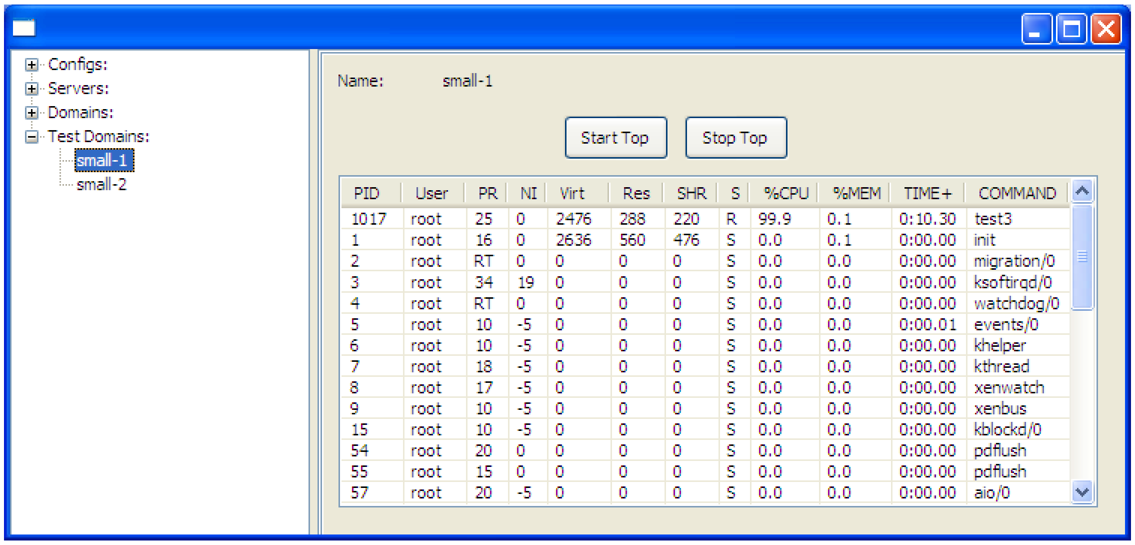
State: Running

CPU Usage: cpu 0/3: 100.0

Figure 10: Domain Composite

6.5 Test Composite

This composite provides the ability to view the processes and their properties running on a virtual machine. Figure 11 shows that test3, the Fibonacci program, is using 99.9% of the CPU.



Name: small-1

PID	User	PR	NI	Virt	Res	SHR	S	%CPU	%MEM	TIME+	COMMAND
1017	root	25	0	2476	288	220	R	99.9	0.1	0:10.30	test3
1	root	16	0	2636	560	476	S	0.0	0.1	0:00.00	init
2	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
4	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
5	root	10	-5	0	0	0	S	0.0	0.0	0:00.01	events/0
6	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	khelper
7	root	18	-5	0	0	0	S	0.0	0.0	0:00.00	kthread
8	root	17	-5	0	0	0	S	0.0	0.0	0:00.00	xenwatch
9	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	xenbus
15	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	kblockd/0
54	root	20	0	0	0	0	S	0.0	0.0	0:00.00	pdflush
55	root	15	0	0	0	0	S	0.0	0.0	0:00.00	pdflush
57	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	aio/0

Figure 11: Test Composite

7. Agents

7.1 Introduction

There are three agents that make up this system: the server agent, the configuration agent and the virtual agent. These agents provide information and are able to execute commands on the machines on which they are running. All of the agents use the Java Message Service (JMS) to publish information and to subscribe to commands from the GUI and events from other agents.

7.2 Server Agent

The server agent is made up of several components used for monitoring, managing and migrating virtual machines. The server agent is run in Dom0 of all of the hosting machines.

Heartbeat

Every few seconds the agent publishes a heartbeat on the *HeartBeat* message topic. This heartbeat contains the server name, individual CPU load as well as the virtual machines and their statuses. All other server agents on all other systems (running as Dom0) listen for Heartbeat messages.

Watchdog

The Watchdog agent is connected as both a publisher and a subscriber to the *watchdog* message topic. It monitors the CPU, network and disk usage of the physical machine on which it is running. However in our case, migrating a virtual machine to another host will not yield better network or disk performance. If the agent notices that its machine is getting overloaded it will publish a message on the *watchdog* message topic asking for help from other watchdog agents on the network. When other agents receive this message, they determine whether they can take on another virtual machine and if they can, they reply on the *watchdog* message topic. The overloaded machine will take the first offer and start the migration process. To determine if the server is overloaded the agent watches its CPU usage and receives one point for every time unit over 80% and looses a point for every time unit under 80%. If the server makes it to 5 points then action is taken. This heuristic seemed to work well in our testing scenarios. The function that does this computation could be easily replaced by one with a more complex algorithm.

Virtual Manager

The VirtualManager agent is a publisher on the *DomainStatus* message topic and a subscriber on the *DomainControl* message topic. This class is in charge of receiving messages from the GUI and running Xen commands like pause and unpause and then sending the updated state back to the GUI.

7.3 Configuration Agent

The configuration agent runs on the configuration server that is storing the configuration files as well as GNBD file system exporting the LVM partitions to the virtual machines. When the agent is first started it makes sure that all of the partitions are currently being exported by gnbd_export. It then begins publishing heartbeat messages and subscribes to the *Configs* message topic to receive Config messages from the management GUI. The information from the Config message is used by the agent to create the swap and disk partitions for the virtual machine using LVM. Once the partitions are created, they are exported using GNBD. The disk partition is then mounted as a loopback device and debootstrap is used to create a base Ubuntu installation. Debootstrap is a tool provided by Debian that creates a base system by downloading .deb packages and extracting them into a given directory. Once this is complete additional packages are installed like OpenSSH and Java. The agent then proceeds to update the configuration files to reflect the machine name and network information. Then files for the tests and virtual agent are copied to the virtual machine. Once this is complete, the disk image is un-mounted and a broadcast is made over the message bus on the *DomainControl* message topic to let the server agents know that a new GNBD export is available.

HeartBeat

Like the server agent, the configuration agent also has a heartbeat component. This heartbeat reads all of the configuration files in the configs directory and then publishes the information to the management GUI on the *Configs* message topic.

7.4 Virtual Agent

The virtual agent runs inside of each virtual machine, DomU. The agent publishes a heartbeat, on the *virtualmachine* message topic, with just its name to let the GUI know it is there. The agent is able to start tests such as an exponential Fibonacci algorithm or image manipulation that will drive up the load on that virtual machine. To start one of these, the GUI sends a message on the *virtualCommands* message topic containing a list of commands to be run. Upon completion of these commands the client replies on the *virtualReturn* topic with amount of time the commands took to run. The GUI can also request that the agent publish snapshots of the Unix utility top which shows process statuses for that virtual machine.

8. Publish/Subscribe Message Types

There are three interface messages. The GenericMessage class is the top of the message hierarchy and only requires that its children will be serializable. The ControlEvent class is the parent for all of the messages that require action by the recipient and the InfoMessage is the parent for all of the messages that are providing information to the recipient.

Control Events

The CommandEvent message is sent from the Test Manager and the Virtual Client. It contains a list of commands that the receiving virtual machine should run.

The ConfigsEvent message is sent from the GUI to the configuration client. The message contains a mode, currently only *CREATE_DOMAIN*, and an instance of the Config class. The Config class contains the name of the new domain, the memory, the swap and hard disk space it should be allocated.

The serverEvent message is sent from the GUI to a server client. It can do one of three things. The first is to change the interval at which the heartbeats are published. The second is to set the server to not accept migration from other machines in the network. The third is setting the server to accept migration from other machines.

The virtualDomainControl is a message that is sent from the GUI to the virtual client. The message tells the client to either start or stop sending virtualTopMessages to the GUI.

The WatchdogMessage is sent and received by the server client. The three modes of this message are REQUEST_HELP, OFFER_HELP and ACCEPT_HELP. It contains a domainSummary instance as well as the sending server.

The xendomainEvent is a message that is sent from the GUI to the server client. It contains information to alter the state of a domain in addition to initiating the updating of the GNBD imports.

Info Messages

The ConfigsMessage is a message from the configuration client to the GUI. It contains a vector of the configs that are currently present.

The domainMessage is a message that is sent from the server client to the GUI. It contains more specific information about a domain than the heartbeat message through having an instance of the domainSummary class.

Message Types

The HeartbeatMessage is a message that is sent from the server client to the GUI. It contains the interval at which the heartbeats are being sent, an instance of the serverSummary class and whether or not it is currently accepting domains.

The virtualDomainMessage is a message that is sent from the virtual client to the GUI. It contains an instance of the virtualSummary class.

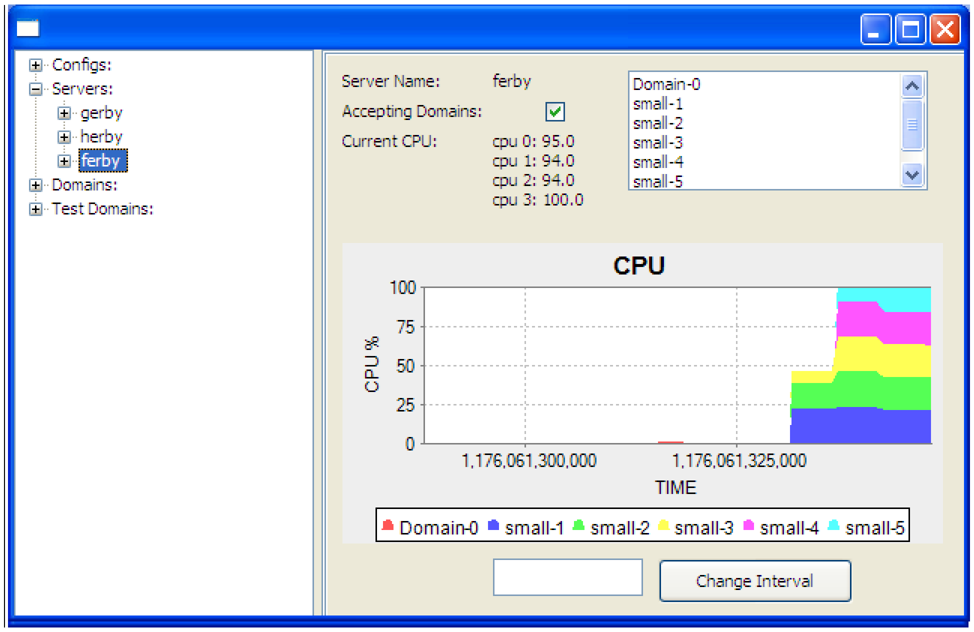
The virtualReturnMessage is a message that is sent from the virtual client to the test manager. It contains an instance of the scheduleItem. It is used to report the time it took to run a specific test.

The virtualTopMessage message is used to forward the output from Unix utility top from the virtual agent to the GUI.

9. Live Migration Example

This example will give a quick run through of what occurs when a machine determines that it is overloaded. Ferby, Herby and Gerby are Xen machines, Derby is the JMQ server and Kerby is the GNBD server. Ferby will have five virtual machines generating load which will take 100% of all four CPUs. Ferby's watchdog agent will broadcast for help and Herby will respond. Ferby will accept the offer and migrate a virtual domain.

1. Five virtual domains are started on Ferby and a high load test application is started on each of them.



Server Name: ferby
Accepting Domains: ✓
Current CPU: cpu 0: 95.0
cpu 1: 94.0
cpu 2: 94.0
cpu 3: 100.0

Domain-0
small-1
small-2
small-3
small-4
small-5

Figure 12: GUI View of Ferby Before Migration

Ferby's watchdog agent realizes that its server is overloaded (all 4 CPUs have greater than 80% load and at least one of which has multiple virtual machines) and that it needs some help. It broadcasts a JMS message to the JMQ server and other watchdog agents that are subscribers receive the message.

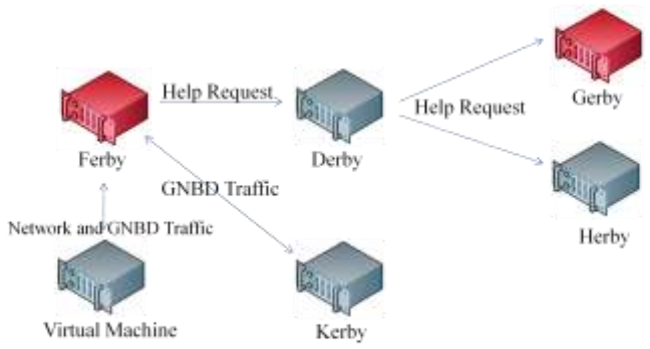


Figure 13: Ferby Asks for Help

2. Gerby’s watchdog agent receives the message and checks its server’s status. It is currently accepting virtual machines but is under too much load so it does not reply. Herby’s watchdog also receives the message and it checks its machine’s status. It is currently accepting virtual machines and is not overloaded. It replies to Ferby with a message offering to help.

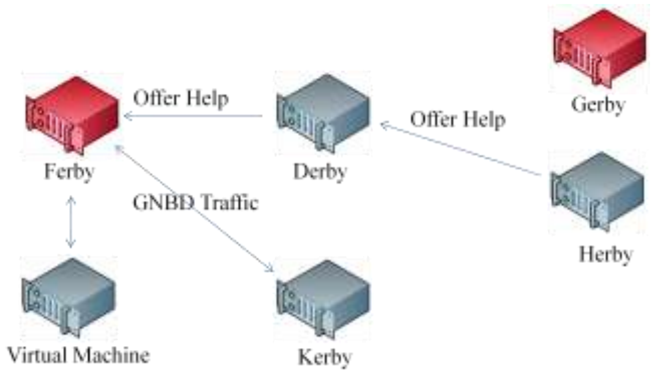


Figure 14: Herby Offers Help

3. Ferby’s watchdog agent receives the help offer from Herby and decides to take Herby up on the offer (Herby was the first machine to respond). So Ferby’s watchdog sends an acknowledgment back to Herby and will decline any other offers of help.

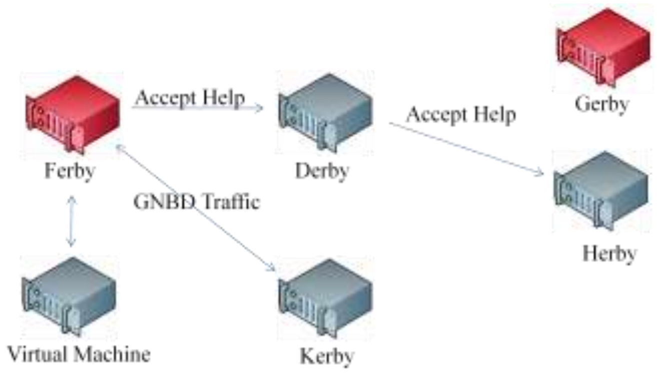


Figure 15: Ferby Accepts Help

4. Shortly after accepting the offer, Ferby's watchdog instance initiates the virtual machine migration by using Xen's live migration interface.

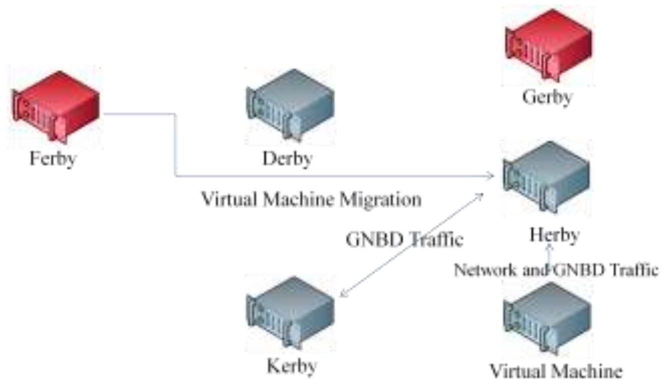
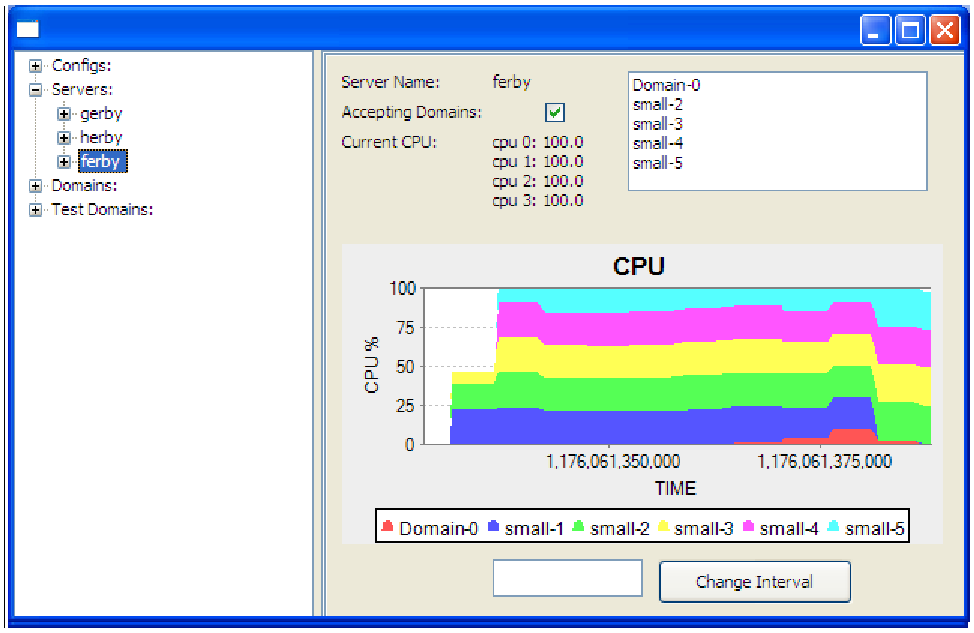


Figure 16: Virtual Machine is Migrated

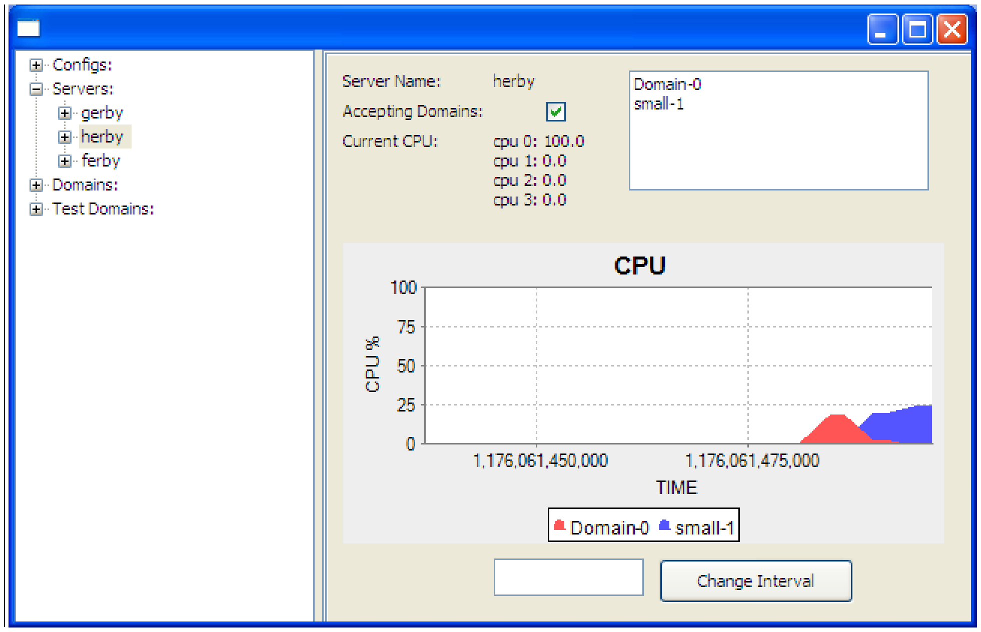
During the migration, CPU usage for Domain 0 on both Ferby and Herby spikes but after migration is complete Domain 0 CPU usage falls as shown in the charts of Figure 17 and Figure 18. As can be seen in Figure 17 and Figure 18 the virtual domain small-1 has been migrated to Herby. In Figure 17, Ferby still has 100% on all CPUs but now each CPU only has a single virtual machine. Since now each virtual CPU has an entire physical CPU to itself the watchdog agent will not attempt to live migrate additional domains. If there were a faster physical machine available, it might be beneficial to migrate virtual domains to it but all of our test machines have the same specifications. After the migration is completed the networking and disk access has changed from being routed through Ferby to Kerby to being routed from Herby to Kerby. The network is rerouted by having the migrated virtual machine send out an unsolicited ARP reply that directs traffic to its new location. In our setup the disk access is moved by simply having Ferby unmount the GNBD partition and having Herby mount it.



Server Name: ferby
Accepting Domains: ✓
Current CPU: cpu 0: 100.0
cpu 1: 100.0
cpu 2: 100.0
cpu 3: 100.0

Domain-0
small-2
small-3
small-4
small-5

Figure 17: GUI View of Ferby After Migration



Server Name: herby
Accepting Domains: ✓
Current CPU: cpu 0: 100.0
cpu 1: 0.0
cpu 2: 0.0
cpu 3: 0.0

Domain-0
small-1

Figure 18: GUI View of Herby After Migration

10. Testing Environment

Test Setup

Virtual Machine Hosts (x4)	
Processor:	Dual Core AMD Opteron Processor 275 (x2)
RAM:	16gb DDR
Hard Drive:	300gb (x2)
Networking:	10/100/1000-Mbps Ethernet port (x2) 10/100-Mbps Ethernet with two external ports
Operating System:	Ubuntu Linux 6.06 Dapper
Xen:	3.0.3

JMQ Server and Configuration/Image Server	
Processor:	Dual Core AMD Opteron Processor 175
RAM:	4gb DDR
Hard Drive:	250gb (x2)
Networking:	10/100/1000-Mbps Ethernet port (x2)
Operating System:	Ubuntu Linux 6.06 Dapper for the Configuration/Image Server and Solaris 10 for the JMQ Server

One of the objectives of this project was to create an automated system that would keep the workload for a set of physical machines as even as possible, by allowing overloaded physical machines on the network to offload virtual machines to underutilized physical machines on the network. To test this we examined the effect migration has on the time taken to complete a set of tasks. After many tests and migration algorithms later we came to the realization that the goal of this project was not to be able to quickly distribute 16 high load virtual machines that were all started at the same time but rather to move the one virtual machine that pushed a physical machine into an overloaded state. It would be possible to calculate how machines would have to be moved to put a given physical machine back into an acceptable state and migrate them all at once to another machine. As the load on a given physical machine grows, the amount of time it takes to migrate a virtual machine increases and it becomes apparent that in order to speed up the process it would be faster to pause all of the virtual machines, do the migration and then resume the virtual machines. This is an unacceptable solution because people using the virtual machines would notice a drop in service.

The system takes approximately 10 seconds to migrate a virtual machine off an overloaded physical machine with a disruption of service lasting in the low hundreds of milliseconds. The majority of the time is used to make sure that the load on the server is going to stay high and that it is not just a spike. This algorithm could be fine-tuned to fit the environment. For example if users are writing and compiling software there will be a lot of spikes so the delay should be longer. Otherwise, if the users are executing long running computationally intensive programs then the delay should be shorter. In future releases an adaptive algorithm could be used to monitor for domains that are usually idle and only occasionally spike in CPU or domains that once load is high it will remain high for a longer period of time. With this additional information better choices could be made about whether to migrate a domain and if so which domain to migrate.

11. Conclusion

When this project was first undertaken, the goal was to create a computing environment that would enable all students to have their own virtual machines. This seemed like a possible idea except for the fact that there was no guarantee that all students assigned to a particular physical machine wouldn't be using it at the same time. Also there would need to be a facility for the administrator to easily add more virtual machines. Although there were complicated system problems, I think that we have accomplished what we set out to do.

12. Future Work

Given more time there are many things that could be improved upon. Once Xen support becomes more stable it would be beneficial to add more operating systems to the image creation tools. On the networking side, it would be nice to be able to easily divide virtual machines into VLANs. Also when Xen is working with Open Solaris it would be interesting to implement a system that makes use of their new virtual network adapters.[11] Further automation of the system by allowing for the "jumpstarting" of the physical machines could be useful. Using DNS mapping or hosts file modification to make connecting to virtual machines more straightforward. As discussed above it would be beneficial to have a migration algorithm that would be able to adapt to different situations. There is also the possible of the algorithm giving priority to certain virtual machines by keeping them on physical machines with less load.

13. References

1. CNET Networks. (2007, February 11). *Intel shows off 80-core processor*. Retrieved February 20, 2007, from news.com: http://news.com.com/Intel+shows+off+80-core+processor/2100-1006_3-6158181.html

2. King, S. T., Dunlap, G. W., & Chen, P. M. (2005). Debugging operating systems with time-traveling virtual machines. *Proceedings of the 2005 Annual USENIX Technical Conference*.

3. Sun Microsystems, Inc. (n.d.). *Java Message Service (JMS)*. (Sun Microsystems) Retrieved from Sun Developer Network: http://java.sun.com/products/jms/

4. Biallas, S. (n.d.). *PearPC - About.* Retrieved March 1, 2007, from PearPC: http://pearpc.sourceforge.net/about.html

5. Wikipedia. (n.d.). *Implementation of virtual processing*. Retrieved from VMware - Wikipedia: http://en.wikipedia.org/wiki/Vmware#Implementation_of_virtual_processing

6. Ben-Yehuda, M., Mason, J. D., Krieger, O., & Xenidis, J. (2006). Xen/IOMMU: Breaking IO in New and Interesting Ways. *Austin Xen Summit.* Austin.

7. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., et al. (2003). Xen and the Art of Virtualization. *ACM Symposium on Operating Systems Principles (SOSP).*

8. Clark, C., Fraser, K., Hand, S., & Hansen, J. G. (2005). Live Migration of Virtual Machines. *NSDI.*

9. Red Hat. (n.d.). *GNBD Project Page*. Retrieved from http://sourceware.org/cluster/gnbd/

10. Red Hat. (n.d.). *GULM Project Page*. (Red Hat) Retrieved from Cluster Project Page: http://sources.redhat.com/cluster/gulm/

11. Sun Microsystems, Inc. (n.d.). *Crossbow: Network Virtualization and Resource Control*. Retrieved from Open Solaris: http://opensolaris.org/os/project/crossbow/

12. Clark, B., Deshane, T., Dow, E., Evanchik, S., Finlayson, M., Herne, J., et al. (2004). Xen and the Art of Repeated Research. *USENIX Annual Technical Conference.* Boston.

13. Quetier, B., Neri, V., & Cappello, F. (2005). Scalability Comparison of 4 Host Virtualization Tools.

14. Standard Performance Evaluation Corporation. (n.d.). *SPECweb99*. Retrieved from Standard Performance Evaluation Corporation: http://www.spec.org/osg/web99/