Technical Report
NWU-CS-04-39
July 20, 2004

Virtuoso: A System For Virtual Machine Marketplaces

Alex Shoykhet Jack Lange Peter Dinda

Abstract

In most existing computing environments, resources such as CPU time, memory, disk space, and network bandwidth are not used to capacity. In future environments, such as computational grids, resources may exist solely for sale. There is no clean way for a resource provider to sell either kind of resource because the current abstractions an owner can provide (and a buyer can request) are (1) too high level, (2) numerous, and (3) complex. Virtuoso is a prototype system that creates a marketplace in which resources can be straightforwardly sold by resource providers to resource buyers. The resource exchange is in the form of a low level virtual machine and virtual network that presents the abstraction of a new raw physical machine on the buyer's network. This report describes the interface and implementation of the Virtuoso system. It is also a user manual for those who wish to try Virtuoso.

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, and EIA-0224449. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

Keywords: virtual machines, distributed computing, economic models of computing

Virtuoso: A System for Virtual Machine Marketplaces

Technical Report NWU-CS-04-39

Alex Shoykhet Jack Lange Peter Dinda
a-shoykhet,jarusl,pdinda@northwestern.edu
Department of Computer Science
Northwestern University

August 8, 2004

Abstract

In most existing computing environments, resources such as CPU time, memory, disk space, and network bandwidth are not used to capacity. In future environments, such as computational grids, resources may exist solely for sale. There is no clean way for a resource provider to sell either kind of resource because the current abstractions an owner can provide (and a buyer can request) are (1) too high level, (2) numerous, and (3) complex. Virtuoso is a prototype system that creates a marketplace in which resources can be straightforwardly sold by resource providers to resource buyers. The resource exchange is in the form of a low level virtual machine and virtual network that presents the abstraction of a new raw physical machine on the buyer's network. This report describes the interface and implementation of the Virtuoso system . It is also a user manual for those who wish to try Virtuoso.

keywords: virtual machines, distributed computing, economic models of computing

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, and EIA-0224449. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).

Contents

1 Introduction

In most computing environments, resources such as CPU time, memory, disk space, and network bandwidth are not used to capacity. Those resources are wasted and the owner of the resources has no way to profit from the resources that he does not use. In the same vein, a provider who wishes to install resources explicitly for the purpose of sale finds the process daunting. Potential buyers will have great difficulty finding collections of machines and other resources that provide the software and hardware that they need. Distributed computing is currently a very high friction endeavor.

The problem is that the abstraction is too high level. Current models in distributed computation and cycle stealing, such as RPC, distributed shared memory, processes, and threads are enmeshed in software complexity: operating systems, versions, shared libraries, software installations, need for root level access,etc. Having a buyer's requirements along these dimensions match with what a provider has on offer is rare.

In contrast, Virtuoso's abstraction, a virtual machine [4, 1, 14, 9, 8, 11] (VM), is very low level and simple. The buyer receives a remote machine, configured with the CPU, memory, and disk resources he desires, that is indistinguishable from an actual physical machine. The buyer can install whatever software he needs on the machine, including whole operating systems. A virtual network, VNET, ties the buyer's VMs together efficiently and makes them appear, regardless of wherever they currently are, to be directly connected to the buyer's LAN. VNET is evolving into an adaptive virtual network that can use inter-VM traffic pattern inference, VM migration, overlay topology and routing rule manipulation, and resource reservations to optimize the performance of a distributed or parallel application running in a buyer's VMs. A detailed argument for the virtual machine / virtual network model of distributed computing is available elsewhere [3], as well as detailed information on VNET and other elements of the system [13, 5, 6, 2, 12]. This report thoroughly describes the implementation of the Virtuoso component of the system.

The goal of Virtuoso is to create a marketplace for virtual machines, making it straightforward for a resource provider to sell resources packaged as VMs to buyers. From the buyer perspective, Virtuoso makes it easy to create VMs on the provider's physical machines that are then owned and controlled by the buyer. The buyer specifies the constraints for his virtual machine in terms of service rates ($/bogomip, $/GB-hour, etc). The resource provider registers his physical machine and specifies service rates for its resources. Virtuoso match buyers and providers based on their constraints.

The experience for the buyer is as close as possible to the purchase and use of a real physical machine. We refer to it as *The Dell Model*, as it approximates the experience of using the web site of a major computer retailer, such as Dell Computer, to order a computer. In Virtuoso, the computer arrives virtually, through a web-based console display, remote devices, and the virtual network as described above. The typical sequence of steps that a buyer goes through to start using a new physical machine today is: shopping, setting up, use, and maintenance. Virtual machines in Virtuoso also go through a life cycle: registration, storage, setting up, use, and maintenance. The use phase is divided into periods of running the VM, suspending the VM, and migrating the VM between providers in a search for improved performance and cost.

We begin by explaining the high-level architecture of Virtuoso (Section 2. Next, we describe how to install Virtuoso (Section 3), the lifecycle of a VM in Virtuoso (Section 4), and how to use it from a buyer and provider perspective (Section 5). This is followed by a detailed discussion of the software targeted at those who intend to modify or extend it (Section 6), and a discussion and initial performance results for VM migration (Section 7). Finally, we discuss in-progress extensions to the system (Section 8).

2 Components

As shown in Figure 1, there are three roles in Virtuoso: buyer, front-end, and provider. Each of these roles has a corresponding software installation. We are currently integrating Virtuoso (which is described here)

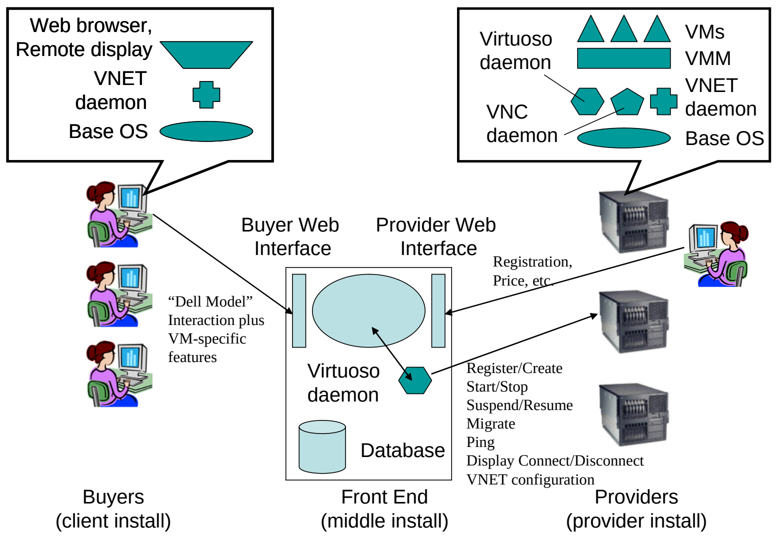


Figure 1: A high-level view of Virtuoso and related components.

and VNET (which is described elsewhere [13]). For this reason and for simplicity, we have included VNET in the figure and in the following discussion.

The buyer, front-end, and provider software works together to provide the buyer of a VM with the following web-based options for their machine: registering, storing, starting, viewing, stopping, suspending, resuming, and migrating. This functionality is achieved through communication between the front-end and the provider systems. The VNET integration will allow the buyer to control and view his virtual network and the physical network from the front-end.

For the provider, the front-end provides a web interface through which he can register machines and other resources on which he is willing to instantiate VMs, and the prices he will charge. The front-end system also serves as the "scoreboard" for the marketplace. Both providers and buyers can view the status of the market and react accordingly. and can manage and view their accounts.

The front-end software consists of buyer and provider CGI web interface code talking to a common core, a web server, a database server, and a Virtuoso daemon for talking to providers.

The buyer software consists minimally of a web browser with Java support and a VNET daemon (when integration is complete). By default, the VM's console is displayed using a automatically downloaded VNC applet. Additional display software, such as a VM-specific remote desktop viewer or an ssh client can also be installed for better display performance.

The provider software consists of a virtual machine monitor (VMM), a Virtuoso daemon, a VNC daemon, and a VNET daemon. The Virtuoso daemon manages VMs running under the VMM, and coordinates with Virtuoso daemons on other machines to control and migrate VMs. The VNC daemon provides VMM-independent remote console display of VMs. The VNET daemon coordinates with VNET daemons on other machines to provides efficient network connectivity to the VMs.

3 Installing Virtuoso

This section describes what is needed to run Virtuoso, where to get it, and how to install it.

3.1 Requirements

Most of Virtuoso is written in Perl—the current version was developed on Perl v5.8.0 but should run on any Perl 5. We expect that it is running on Linux. We test on a “everything” installation of Red Hat Linux 9.0.

The expectation is that there is a **Virtuoso** user on the front-end and provider systems. This is currently a requirement for using the migration system.

Virtuoso is provided with no warranties or guarantees of any kind and we can not provide support for it at this time.

3.1.1 Buyer

The buyer requires a modern web browser that supports Java applets. We have successfully used Microsoft Internet Explorer and Mozilla.

Internet Explorer’s default option is to prevent Java applets from accessing any address other than the one from which the applet was served from. In Virtuoso, the front-end machine will serve a display applet which will try to connect to whichever provider machine the user’s VM is running on. This means that the default behavior is to refuse to connect to any provider machine that isn’t also the front-end machine[1]. A fix to this is to edit the Java permission settings: go to Tools -> Internet Options -> Security -> Custom Level -> Java Permissions -> Custom -> Java Custom Settings -> Edit Permissions -> Unsigned content -> Access To All Network Addresses -> Enable -> Click OK a few times.

In the final integration of Virtuoso and VNET, the buyer will also need root access to his machine to install VNET.

We plan to provide bootable CDs for the buyer and provider software at some point.

3.1.2 Front-end

The bulk of the Perl code in Virtuoso consists of scripts that provide the Web CGI interface of Virtuoso. Virtuoso requires a web server (we use the Apache server that comes with RH 9 for testing) and you must have CGI enabled for the directory in which you install the front-end code.

Virtuoso is able to run on an Oracle database or on an internal text file database. We typically run it on Oracle 9.2 Enterprise Edition. It will probably run on other databases since our schema, queries, and updates are simple, and we use DBI.

3.1.3 Provider

The virtual machine monitor that Virtuoso uses is VMWare GSX Server 2.5.1 build-5336. Any 2.x version of this product should work. We have tried to minimize dependencies on VMWare. However, if you want to port Virtuoso to some other VMM, be aware that it is essential that it have a scriptable interface. At the present time VMWare’s workstation product does not.

3.2 Getting Virtuoso

If you have access to the Virtuoso repository, use CVS to check out the module **virtuoso-development**. The subdirectories **client_install**, **middle_install**, and **provider_install** contain the buyer, front-end, and provider software.

A public release of Virtuoso is planned. It will be available on virtuoso.cs.northwestern.edu.

[1]The Java error message is an rfb.Proto error and says ConnectionRefused

3.3 Installing the buyer package

At the present time, there is no buyer package. Once our integration with VNET is complete, information about the buyer package will be available here. Generally, once the front-end is installed and working, it will be possible to download the buyer package from the initial web page.

In the repository, the **client_install** subdirectory will contain the buyer package.

3.4 Installing the front-end package

There is only one front-end in a Virtuoso installation for a given set of providers and it should remain this way because the front-end keeps a database of all the machines in the system and assigns global unique identifiers. Having **n** front-ends would mean having **n** sets of disjoint providers.

The following is sample output from the installation process, which is run from the **middle_install** subdirectory.

The first script that is run is **middle_make_package.pl**. This script flushes the database contents to an **initialize_tables.pl** file and then makes a gzipped tarball out of the **public_html** and **server_interface** directories.

You have now packaged up the current front-end software and created a custom installer for it, **middle_install.pl**. The next step is to copy ./middle_install.tar.gz into the CGI directory (∼virtuoso/public_html, for example), then run **middle_install.pl**:

```
[shoykhet@behemoth public_html]$ ./middle_install.pl
What is the perl program path that all the Virtuoso files will use?
Press ENTER if it's "#!/usr/bin/perl -w".

What is the ip address for this machine? Press ENTER if it's 129.105.XXX.YYY .

On what port do you want the migration server to listen?
Press ENTER if it's port 7999

What is the base port number on which you want migration servers
  to be spun off.
If you enter 10000, then migration_server.pl processes will be
started on ports 10000 and up.
 Press ENTER if it's port 10000

Do you wish to use a text-based database or an
  Oracle database (enter 'TEXT' or 'ORACLE')?
ORACLE

What is a user name on which Virtuoso can access
  the database? Press ENTER if it's 'Virtuoso'
shoykhet

What is the password for this user name?
  Press ENTER if it's 'Virtuoso'
XXXXXXXX
```

```
What is the oracle home directory? Press ENTER
   if it’s ’/home/oracle/product/9.2.0’

What is the oracle base directory? Press ENTER
   if it’s ’/home/oracle/product/9.2.0/oraInventory’

What is the oracle sid? Press ENTER if it’s ’VIRTUOSO’
GIS

Updating files in
 /usr/home/shoykhet/public_html/server_interface/
[....]
Creating tables.
Initializing tables from initialize_tables.pl.
```

Possible reasons for wanting to reinstall the front-end on the same host include:

- The codebase has been updated.
- You want to reconfigure the migration system.

It is often easiest to do development in the virtuoso-development subdirectories and then use the installer scripts to push your changes out to an install directory for testing. This can be done with the client and provider software as well

Possible reasons for wanting to install the front-end on a different host include:

- You would like to move the front-end to another machine.[2]
- You would like to start a disjoint marketplace of virtual machines.[3]

3.5 Installing the provider package

Once your front-end is up and running, you will be able to download the provider package from it, following the **Setup Distribution Network** link on the main page. Download both the installation script and the gzipped tarball into the directory from which the server is going to run. You can also fetch a copy of a provider package from http://virtuoso.cs.northwestern.edu, the home page for the Virtuoso project.

The following is sample output of the installation script, **provider_install.pl**.[4]

```
-bash-2.05b$ ./provider_install.pl
What is the perl program path that all the Virtuoso
  files will use?
Press ENTER if it’s "#!/usr/bin/perl -w".
```

[2] Currently there is no system in place to update the individual providers on the new location of the front-end. Therefore, they will not be able to log their operations to the **event_logger** running on the front-end. A simple fix would be to send the new location to every provider in the database. In addition, there is no check to make sure that there are no connections left open between the provider system and the front-end. For these reasons, be sure that no one is using the system at the time of movement.

[3] If this is the case, then you should clear the contents of **initialize_tables.pl** so that the system starts with a fresh set of database tables.

[4] The entire provider system is a set of Perl scripts, except for the make_disk executable. This file is run whenever a new machine is going to be created from scratch. You may want to recompile **make_disk.c** to be sure that it is compatible with your kernel. Also, in directory **vnc_mod** is the file **vncpasswd.c** which sets the vnc password to a given input—run make to recompile it.

```
What is the vmware program path for this computer?
  Press ENTER if it’s /usr/bin/vmware.

What is the ip address for this machine?
  Press ENTER if it’s 10.10.10.206 .
129.105.XXX.ZZZ

On what port do you want the Virtuoso server
  to listen? Press ENTER if it’s port 8000

On what port do you want vnc sessions to be started?
  Press ENTER if it’s port 8001

What is the maximum number of vnc sessions
  that you will permit to be started.
Press ENTER if it’s 20 (minimum is 20)

Updating files in /home/virt-local/virtuoso_server/
[....]
```

3.6 Starting the front-end

The front-end is composed of a set of CGI scripts and daemons. To bring the system to full functionality, the database, the web server, and the following daemons need to be running:

public_html/ storage_update.pl running_update.pl[5] price_average.pl[6]

server_interface/ migration_agent.pl

vm_logger/ event_logger.pl [7]

3.7 Starting the provider system.

To bring the provider system to full functionality, **virtuoso.pl** needs to be running.

4 Lifecycle of a machine and versioning

In the Virtuoso system, a given virtual machine goes through a natural lifecycle, as shown in Figure 2. The states that comprise this cycle are controlled carefully and only a legal state is allowed to be reached from any given state. The system keeps track of the progress of the machine through the lifecycle by keeping a version number. If all of the versions of a machine were to be collected then one could follow the progress the machine’s contents over its lifetime. The machine begins in the **registered** state and moves to the **stored**

[5] These daemons update the account balance of buyers for their running and storage costs. You may pass in a period at which these are updated, the default is 60 seconds. For a full description see Section 6.1.9.

[6] If this script is not run, then the marketplace averages of the configurations will not be updated. Thus the script’s job is to keep the average prices of the configurations fresh.

[7] If **event_logger.pl** is run then all events in both the front-end and provider systems will be logged to one log file. If it is not run, then the events will be logged locally. For a full description see Section 6.1.10.

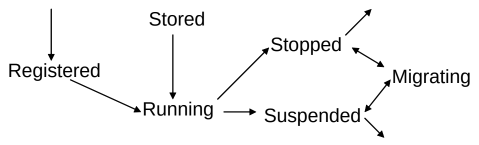


Figure 2: The states of a VM in Virtuoso.

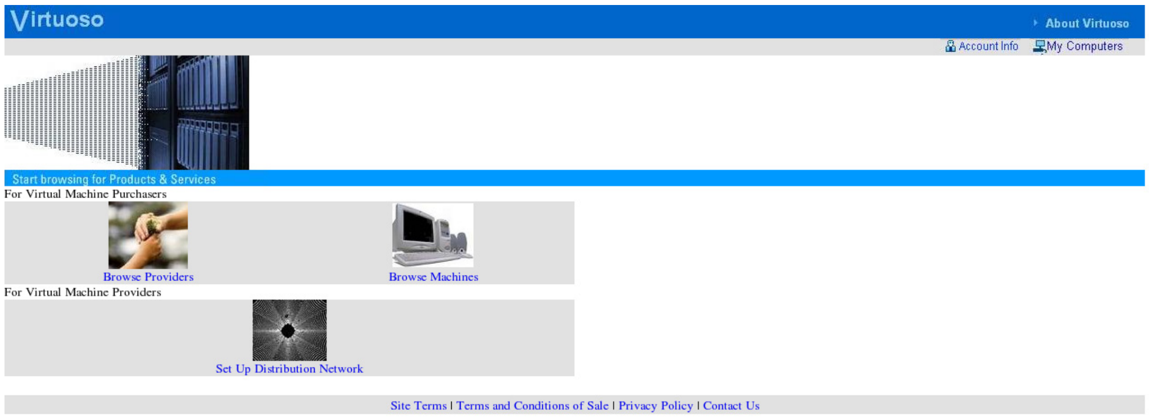

Virtuoso
About Virtuoso
Account Info
My Computers
Start browsing for Products & Services
For Virtual Machine Purchasers
Browse Providers
Browse Machines
For Virtual Machine Providers
Set Up Distribution Network
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 3: The Virtuoso home page.

state. When this happens, its version increases by 1. The machine then goes through a series of states where it is **running** and then **stopped** or **suspended**. Any time that a machine is stopped or suspended, its version increases by 1. Also, a machine goes through a **migrating** state in its life time—at the end of the state its version increases by 1.

The versions of the machine are consistent between the front-end and the provider systems.

5 Examples of using Virtuoso

The following are examples of using the Virtuoso system. The examples are comprehensive in the sense that they demonstrate every function exposed by the Virtuoso front-end to the user. *The technical details of how the functionality is implemented are saved for Section 6.*

Figure 3 shows the Virtuoso home page. Notice that even without logging in, a provider can download the software to join the system (**Setup Distribution Network**).

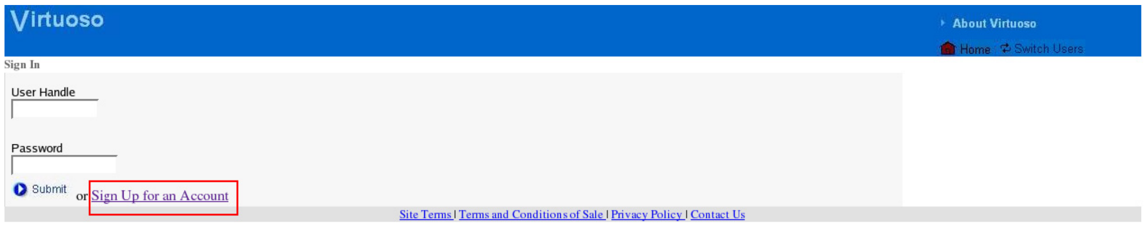
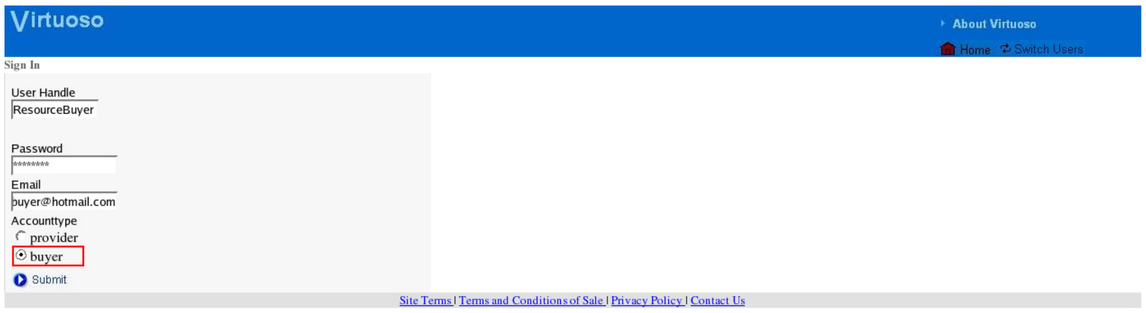

Virtuoso
About Virtuoso
Home
Switch Users
Sign In
User Handle
Password
Submit
or Sign Up for an Account
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 4: The Virtuoso login screen.


Virtuoso
About Virtuoso
Home
Switch Users

Sign In

User Handle
ResourceBuyer

Password
****

Email
buyer@hotmail.com

Accounttype
provider [✓ buyer]

Submit
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 5: Creating a buyer account.

Most of the interface is specific to either a provider user or a buyer user. That is, if a user is logged in as a provider, then they have access to different functions than if they are logged in as a buyer. Any action in the system that is attempted against the access rights of a user type is not permitted. Figure 4 is a snapshot of the log in page—to create a new account click on **Sign up for an account**. If the account under which the user logs in is a provider account, they will have the functionality of a provider. If the account under which the user logs in is a buyer account, they will have the functionality of a buyer.

5.1 Buyer example

This section contains a thorough example of how a buyer can use the full functionality of the Virtuoso system. The description also shows some of the logic behind how the system works.

5.1.1 User registration

Figure 5 shows that when a user wants to create a new account, they have the option of either making it a provider account or a buyer account. The **primary key** that the database uses to identify users is the user name and each user name is unique in the system.

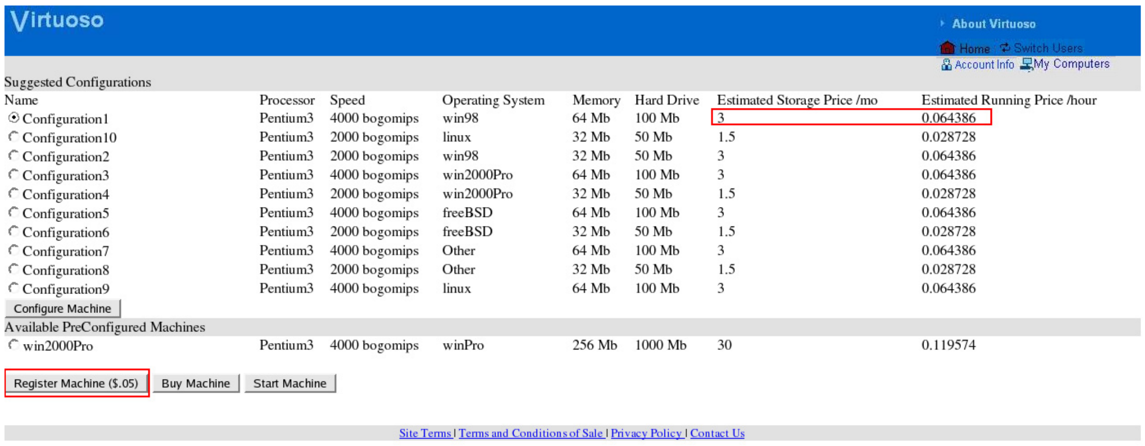
5.1.2 Registering a machine

When a user has logged in as a buyer, he will want to begin creating machines. The first step in the creation of a machine is registering a configuration. To **register** a machine means that Virtuoso will make a placeholder for the machine in the database. The machine does not exist yet on any provider machine. The idea behind this is that the user will want to plan out the specifications of a set of machines before he actually commits to creating them. Figure 6 shows the **browse machines** page in which are listed some standard configurations.

The prices listed with the configurations are market averages for the resources that each configuration is registered with. In Figure 7 the user has clicked on **edit configuration** to use the option of configuring the machine more precisely to their specification.

Figure 8 shows the confirmation screen shown when a machine is registered. The reason that there are two prices listed for storage and two prices listed for running cost is because the machine doesn't exist yet. Thus, the user is not actually being billed for their storage and running costs but has only registered a bid price for the configuration that has been registered.

Once the machine is registered, the buyer can view his machines with the **my computers** function. Figure 9 shows that for each state of the machine, there are corresponding actions that the user can take. In this example, the state is registered and so the user can either **search for providers** or **edit the configuration**.



Suggested Configurations

Name	Processor	Speed	Operating System	Memory	Hard Drive	Estimated Storage Price /mo	Estimated Running Price /hour
[✓ Configuration1]	Pentium3	4000 bogomips	win98	64 Mb	100 Mb	3	0.064386
Configuration10	Pentium3	2000 bogomips	linux	32 Mb	50 Mb	1.5	0.028728
Configuration2	Pentium3	2000 bogomips	win98	32 Mb	50 Mb	3	0.064386
Configuration3	Pentium3	4000 bogomips	win2000Pro	64 Mb	100 Mb	3	0.064386
Configuration4	Pentium3	2000 bogomips	win2000Pro	32 Mb	50 Mb	1.5	0.028728
Configuration5	Pentium3	4000 bogomips	freeBSD	64 Mb	100 Mb	3	0.064386
Configuration6	Pentium3	2000 bogomips	freeBSD	32 Mb	50 Mb	1.5	0.028728
Configuration7	Pentium3	4000 bogomips	Other	64 Mb	100 Mb	3	0.064386
Configuration8	Pentium3	2000 bogomips	Other	32 Mb	50 Mb	1.5	0.028728
Configuration9	Pentium3	4000 bogomips	linux	64 Mb	100 Mb	3	0.064386

Configure Machine

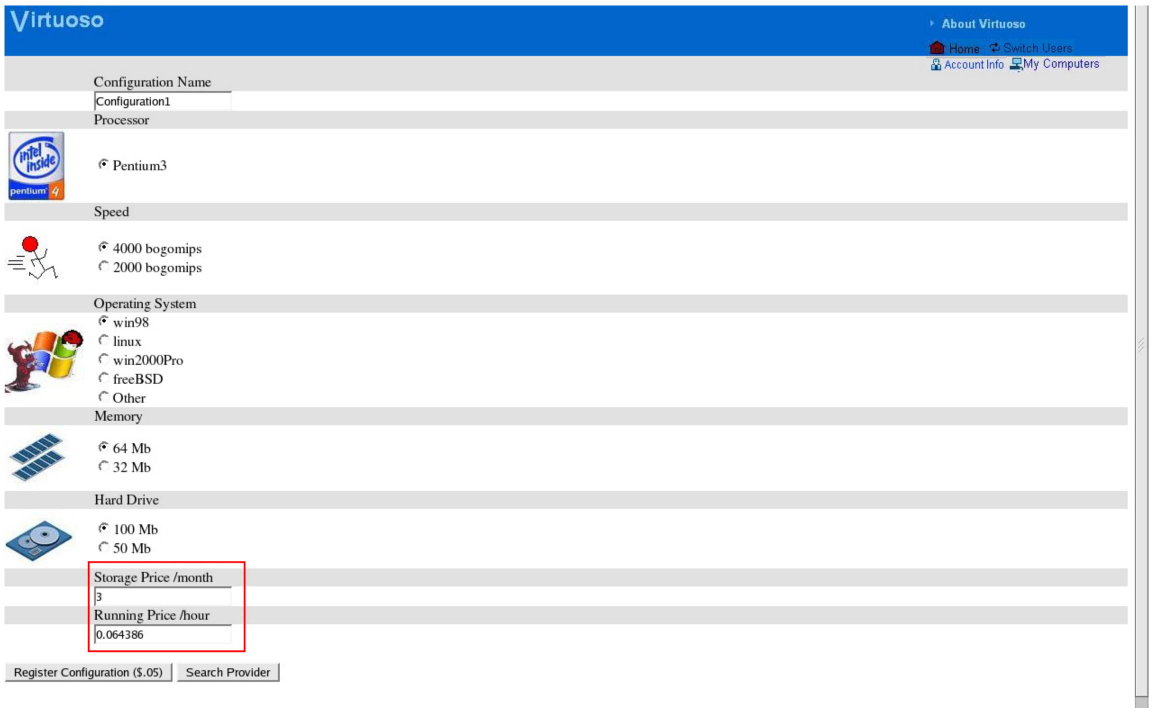
Available PreConfigured Machines

win2000Pro	Pentium3	4000 bogomips	winPro	256 Mb	1000 Mb	30	0.119574

Register Machine ($.05) | Buy Machine | Start Machine

Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 6: Choosing a configuration.



Configuration Name: Configuration1

Processor: [✓ Pentium3]

Speed: [✓ 4000 bogomips] 2000 bogomips

Operating System: [✓ win98] linux win2000Pro freeBSD Other

Memory: [✓ 64 Mb] 32 Mb

Hard Drive: [✓ 100 Mb] 50 Mb

Storage Price /month: 3

Running Price /hour: 0.064386

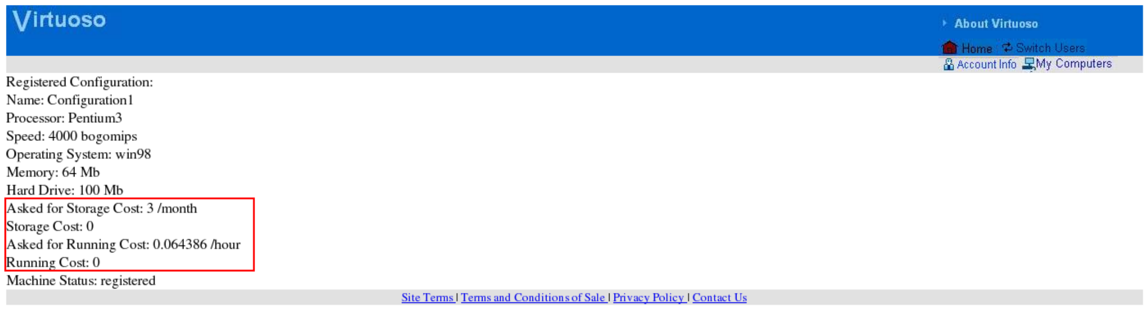
Register Configuration ($.05) | Search Provider

Figure 7: Configuring a machine.

This relationship is held in a relationship table along with all of the possible states of a machine and their possible resulting states. Refer to Section 4 for a full description of what these relationships are.

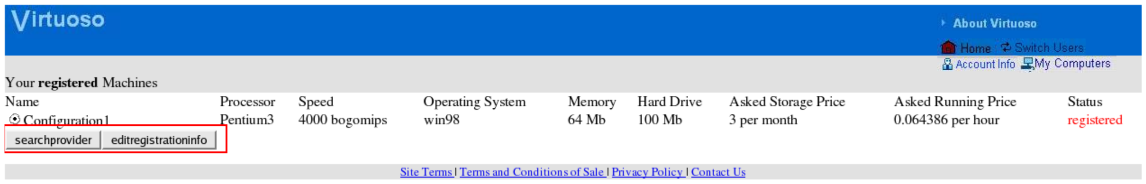
When a user first registers a machine, they have the option of either registering a completely blank set of resources or registering a preconfigured machine. The difference is that, in the latter case (Figure 10), the operating system is already installed.[8]

[8]The major reason for having preconfigured machines is that, in the current system, it is difficult for the user to install a new operating system—since they do not have access to the machine's CD-ROM drive. In later implementations of Virtuoso, the user will have the ability to use their local CD-ROM as if it were the virtual machine's own CD-ROM.


Virtuoso
About Virtuoso
Home
Switch Users
Account Info
My Computers
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Registered Configuration:
Name: Configuration1
Processor: Pentium3
Speed: 4000 bogomips
Operating System: win98
Memory: 64 Mb
Hard Drive: 100 Mb
Asked for Storage Cost: 3 /month
Storage Cost: 0
Asked for Running Cost: 0.064386 /hour
Running Cost: 0
Machine Status: registered

Figure 8: The machine is registered.



Your **registered** Machines

Name	Processor	Speed	Operating System	Memory	Hard Drive	Asked Storage Price	Asked Running Price	Status
● Configuration1	Pentium3	4000 bogomips	win98	64 Mb	100 Mb	3 per month	0.064386 per hour	registered

Figure 9: Buyer controlling machines.

5.1.3 Searching for a resource provider

Once a machine is registered, the user will want to find a provider to host the machine. The **search provider** function matches the resources that the user has asked for and the price that they requested with the providers in the system. Figure 11 shows the results of one such search. If a provider cannot match the buyer's specification, then that provider is not listed as a possible host for the virtual machine.

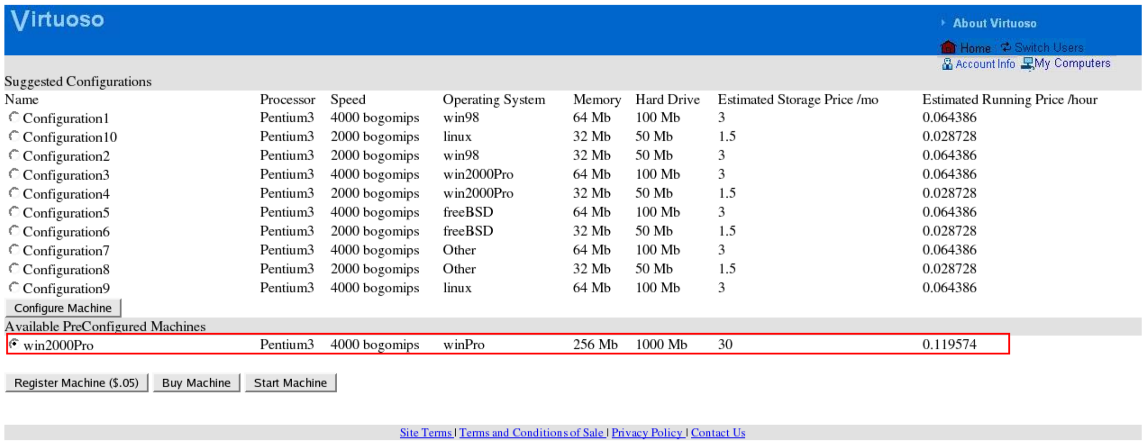
After the user has picked a provider and the machine has been stored, the user can control the machine through the available control options. Figure 12 shows that editing the configuration is no longer an option. This is because the machine now exists on a physical machine (in storage) and the amount of resources it uses cannot change. Also, the user is not able to change the cost of running the machine—unless they migrate to a new provider. Thus, the price is now fixed.

Figure 13 shows the user editing a machine that is in the **registered** state. This is where the user can change the desired amount of memory, disk space, and processor speed with much higher precision than is available in the initial purchase interaction.

The major reason for editing the configuration is to tweak the asked price of the machine's resources. Notice that both the storage price and the running price are expressed as rates. This is an important place where Virtuoso departs from a traditional computer purchasing model. The virtual machine is a service, not an artifact, thus it is priced according to usage, not as a single entity.

As Figure 14 shows, it is typically easy to get the asked price of the machine down low enough so that no provider is able to accommodate the bid.[9]

[9]We envision a user agent that can interact with the purchasing process to continuously find good prices, optimizing the long term cost of running the VM.



Suggested Configurations

Name	Processor	Speed	Operating System	Memory	Hard Drive	Estimated Storage Price /mo	Estimated Running Price /hour
Configuration1	Pentium3	4000 bogomips	win98	64 Mb	100 Mb	3	0.064386
Configuration10	Pentium3	2000 bogomips	linux	32 Mb	50 Mb	1.5	0.028728
Configuration2	Pentium3	2000 bogomips	win98	32 Mb	50 Mb	3	0.064386
Configuration3	Pentium3	4000 bogomips	win2000Pro	64 Mb	100 Mb	3	0.064386
Configuration4	Pentium3	2000 bogomips	win2000Pro	32 Mb	50 Mb	1.5	0.028728
Configuration5	Pentium3	4000 bogomips	freeBSD	64 Mb	100 Mb	3	0.064386
Configuration6	Pentium3	2000 bogomips	freeBSD	32 Mb	50 Mb	1.5	0.028728
Configuration7	Pentium3	4000 bogomips	Other	64 Mb	100 Mb	3	0.064386
Configuration8	Pentium3	2000 bogomips	Other	32 Mb	50 Mb	1.5	0.028728
Configuration9	Pentium3	4000 bogomips	linux	64 Mb	100 Mb	3	0.064386

Configure Machine

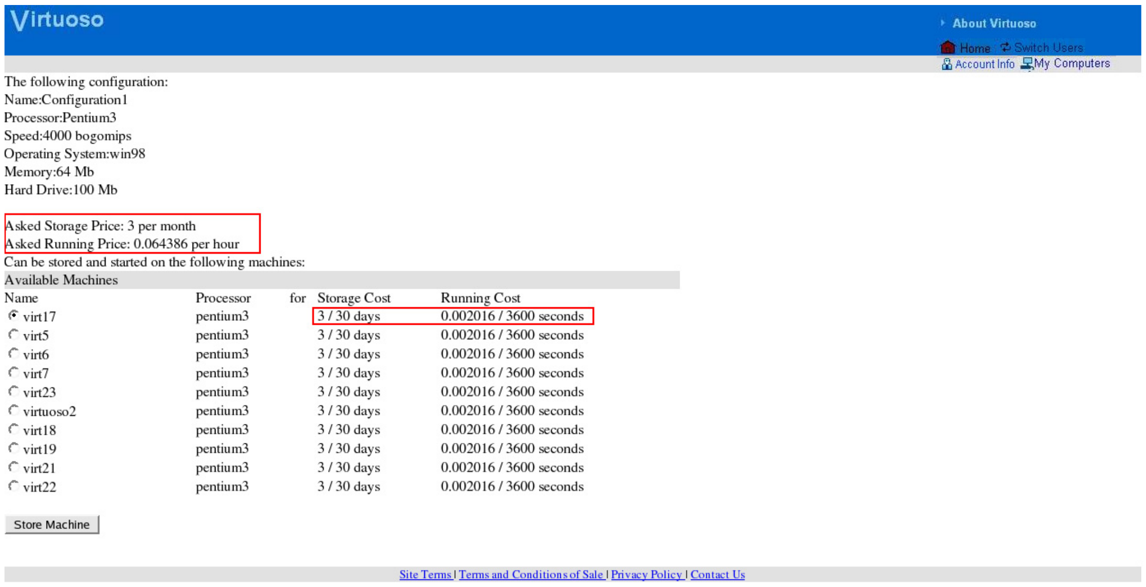
Available PreConfigured Machines

[✓] win2000Pro	Pentium3	4000 bogomips	winPro	256 Mb	1000 Mb	30	0.119574

Register Machine ($.05) | Buy Machine | Start Machine

Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 10: Buyer registering a preconfigured machine.



The following configuration:
Name:Configuration1
Processor:Pentium3
Speed:4000 bogomips
Operating System:win98
Memory:64 Mb
Hard Drive:100 Mb

Asked Storage Price: 3 per month
Asked Running Price: 0.064386 per hour
Can be stored and started on the following machines:

Available Machines

Name	Processor	for	Storage Cost	Running Cost
[✓] virt17	pentium3		3 / 30 days	0.002016 / 3600 seconds
virt5	pentium3		3 / 30 days	0.002016 / 3600 seconds
virt6	pentium3		3 / 30 days	0.002016 / 3600 seconds
virt7	pentium3		3 / 30 days	0.002016 / 3600 seconds
virt23	pentium3		3 / 30 days	0.002016 / 3600 seconds
virtuoso2	pentium3		3 / 30 days	0.002016 / 3600 seconds
virt18	pentium3		3 / 30 days	0.002016 / 3600 seconds
virt19	pentium3		3 / 30 days	0.002016 / 3600 seconds
virt21	pentium3		3 / 30 days	0.002016 / 3600 seconds
virt22	pentium3		3 / 30 days	0.002016 / 3600 seconds

Store Machine

Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

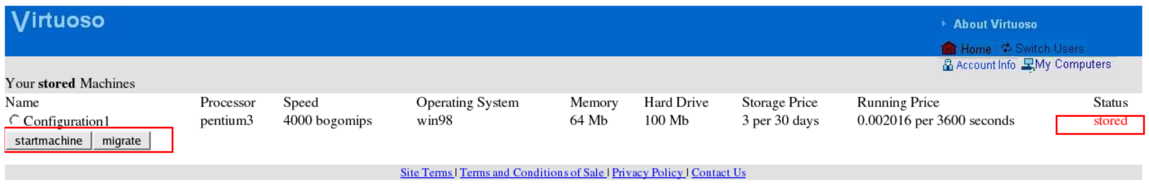
Figure 11: Buyer choosing a provider for the registered machine.

5.1.4 Starting/stopping/suspending/resuming a machine

From the **my computers** interface, the user is able to start the machine that they have stored. Figure 15 shows the VNC login screen that comes up when the user wants to view the console of their machine in their web browser.[10] The password that the VNC server is waiting for is the same password that the buyer used to log into the Virtuoso system. A full description of how the VNC session is managed can be read in Section 6.2.4.

Once the buyer has logged into the VNC session, they can view the machine. Figure 16 shows that, for an unconfigured machine, there is little that the user can do since they cannot install an operating system,

[10]Note that for VNC to work, the web browser must support Java applets and the security settings must permit the VNC applet to make connections to machines other than from where it was downloaded.

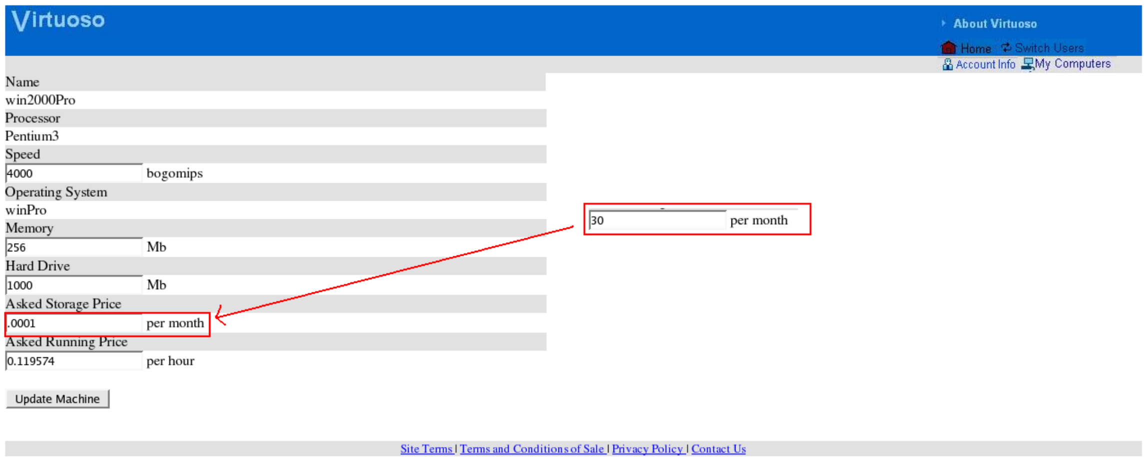


Your **stored** Machines

Name	Processor	Speed	Operating System	Memory	Hard Drive	Storage Price	Running Price	Status
Configuration1	pentium3	4000 bogomips	win98	64 Mb	100 Mb	3 per 30 days	0.002016 per 3600 seconds	stored

startmachine
migrate
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 12: The machine is stored.



Name	
win2000Pro	
Processor	
Pentium3	
Speed	
4000	bogomips
Operating System	
winPro	
Memory	
256	Mb
Hard Drive	
1000	Mb
Asked Storage Price	
.0001	per month
Asked Running Price	
0.119574	per hour

30 per month

Update Machine
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 13: Buyer editing configuration of a registered machine.

as we have not yet integrated remote device support into Virtuoso. Once the development and integration is complete, the user will at this point be able to insert a operating system install CD into their local machine and configure the VM as they desire.

The purpose of the VNC display in the web browser is to make it possible to configure the machine. Most users will not find it fast enough for considerable work. Once the OS has been installed, the user can install faster, OS-specific remote desktop tools, use ssh, etc.

The machine remains running until the buyer asks Virtuoso to either suspend or stop the machine. Figure 17 shows that only the options appropriate for a **running** machine are available to the user.

Figure 18 shows the user viewing a preconfigured machine that already has Windows 2000 installed on it.[11] The Windows Remote Desktop Feature (see Terminal Services) can be used to provide a much faster desktop display. Once VNET is fully integrated, this will become trivial to set up.

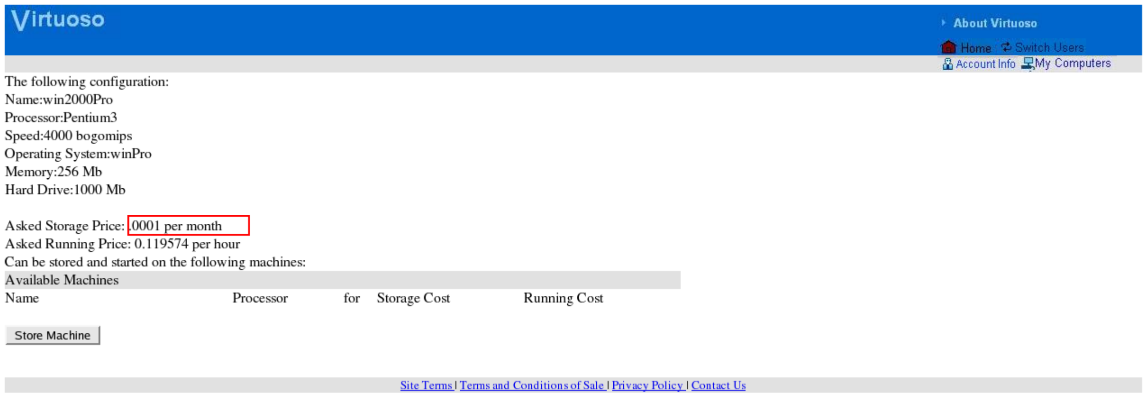
Figure 19 shows how Virtuoso displays the user's machines when different machines are in different states. Again, only the functions appropriate to a given state are permitted by the system.[12]

5.1.5 Migrating a machine

The buyer's ability to move a virtual machine between providers provides the mechanism needed for him to search for a better deal for his machine. This can take the form of looking for a better pricing option or of

[11]Note that VMware allows the user to stop and suspend the machine through VMWare's interface. There is no current way for Virtuoso to pick up that these commands have been made and so, if the user doesn't suspend or stop the machine through the web interface, Virtuoso will assume that the machine is still running.

[12]Note: when suspending a machine, make sure that the machine has loaded completely. VMWare won't let you suspend a machine while the OS is still loading. Virtuoso isn't yet advanced enough to detect this error.



The following configuration:
Name:win2000Pro
Processor:Pentium3
Speed:4000 bogomips
Operating System:winPro
Memory:256 Mb
Hard Drive:1000 Mb

Asked Storage Price: .0001 per month
Asked Running Price: 0.119574 per hour
Can be stored and started on the following machines:

Available Machines

Name	Processor	for	Storage Cost	Running Cost

Store Machine
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 14: Buyer unable to find a provider for their configuration.

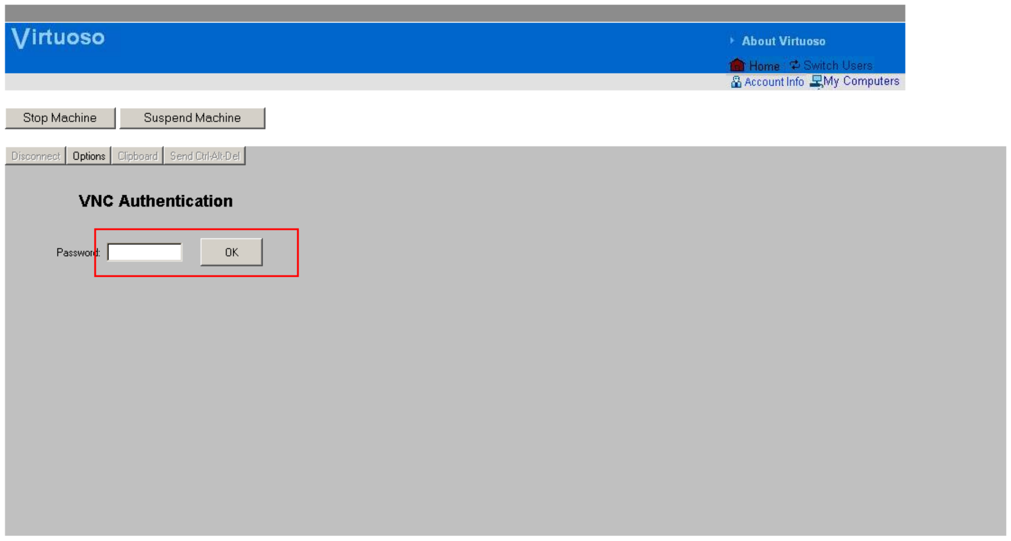


Figure 15: Buyer logging in to view his machine.

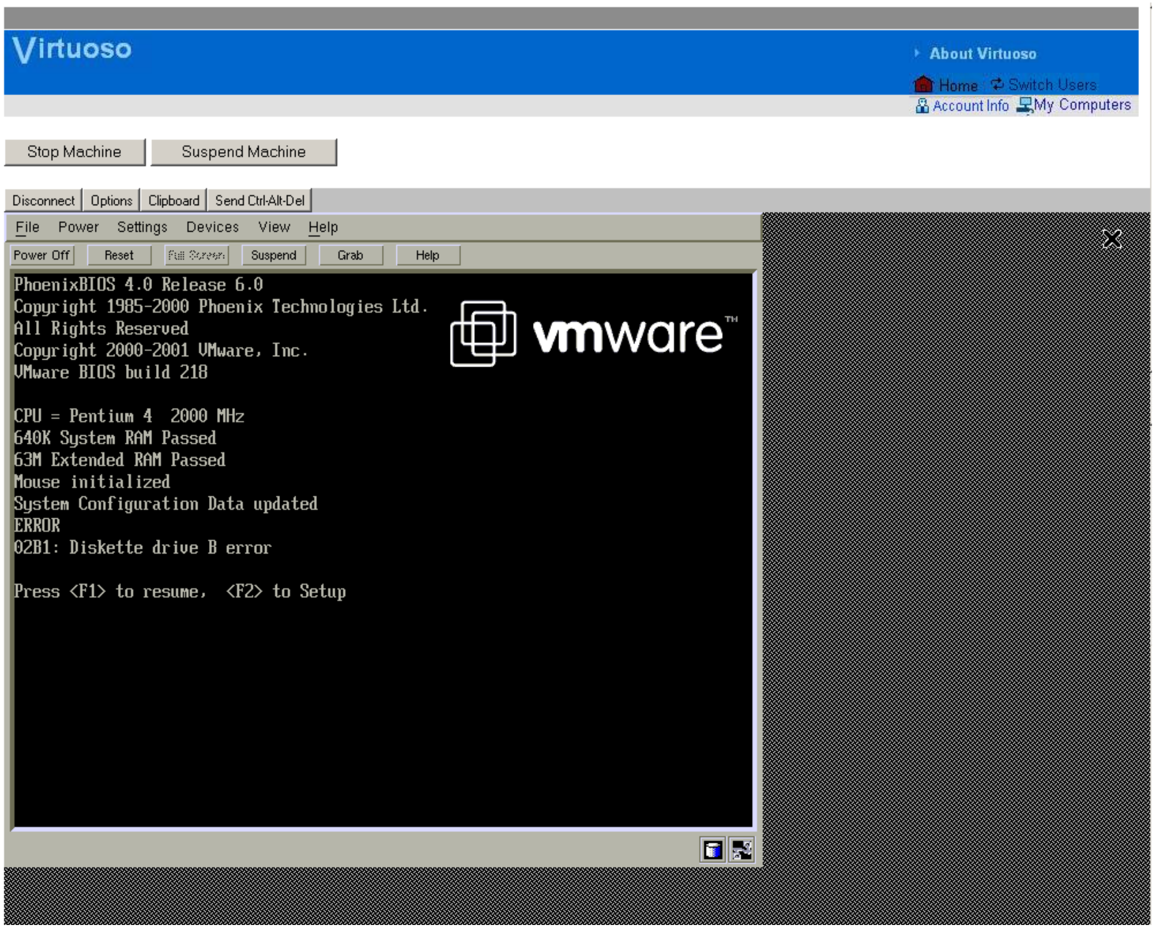
getting better performance.[13]

The search for a provider to migrate to is similar to the search for storing a machine. The Virtuoso system performs a thorough search of its database to come up with a list of providers that match the users specifications. The user has the option of choosing the migration type—this will be the actual physical method with which the machine is transported. Section 7.4 has an initial performance evaluation of the migration system and a comparison of the different types of migration. Figure 20 shows that it is possible for a provider to go offline. In this case, only one provider **virt13** has a server running on their machine and so there are migration options only for that provider.

Once a provider has been chosen, the machine will spend some time in the migrating state. Figure 21 shows that, while the machine is migrating, there are no actions that the user can perform on it.[14]

[13] A research component of this project is examining the scheduling of interactive and batch VMs. There are currently two approaches. The first is user-driven scheduling in which user feedback is used to module VM priority. The second is to schedule the VM as a soft real-time task with a given period and slice.

[14] In this figure there is an option to click on **migrationstatus**. This option is not actually implemented but is rather a placeholder.

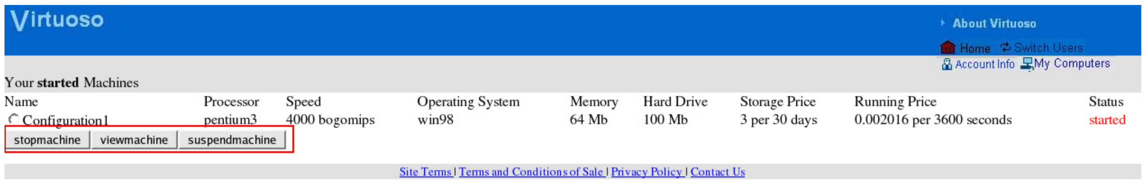


```
PhoenixBIOS 4.0 Release 6.0
Copyright 1985-2000 Phoenix Technologies Ltd.
All Rights Reserved
Copyright 2000-2001 VMware, Inc.
VMware BIOS build 218

CPU = Pentium 4  2000 MHz
640K System RAM Passed
63M Extended RAM Passed
Mouse initialized
System Configuration Data updated
ERROR
02B1: Diskette drive B error

Press <F1> to resume, <F2> to Setup
```

Figure 16: Buyer viewing an unconfigured machine.



Your **started** Machines

Name	Processor	Speed	Operating System	Memory	Hard Drive	Storage Price	Running Price	Status
Configuration1	pentium3	4000 bogomips	win98	64 Mb	100 Mb	3 per 30 days	0.002016 per 3600 seconds	started

Figure 17: Buyer controlling a running machine.

5.1.6 Account management

The user's account is debited continually for storage and for running the machine. When he signed up with a provider, he specified the granularity with which he wished to be charged—that is what is the time period to wait before charging (i.e. 10 days, 30days, etc. for storage and 3600 seconds, 1800 seconds, etc. for running). Figure 22 shows a buyer viewing the cost of all of his machines.

There is no running cost associated with a machine unless the state of the machine is **running**.[15] Fig-

This feature would most likely display an estimated arrival time of the machine based on the current migration speed.

[15]Since Virtuoso wants to ensure that all virtual machines in its system get the memory and speed resources that they registered for, the system does not allow more machines to be stored on a provider than would fill up the memory and speed totals that the provider registered. Thus, the following scenario is possible: a physical machine cannot accept any more virtual machines simply

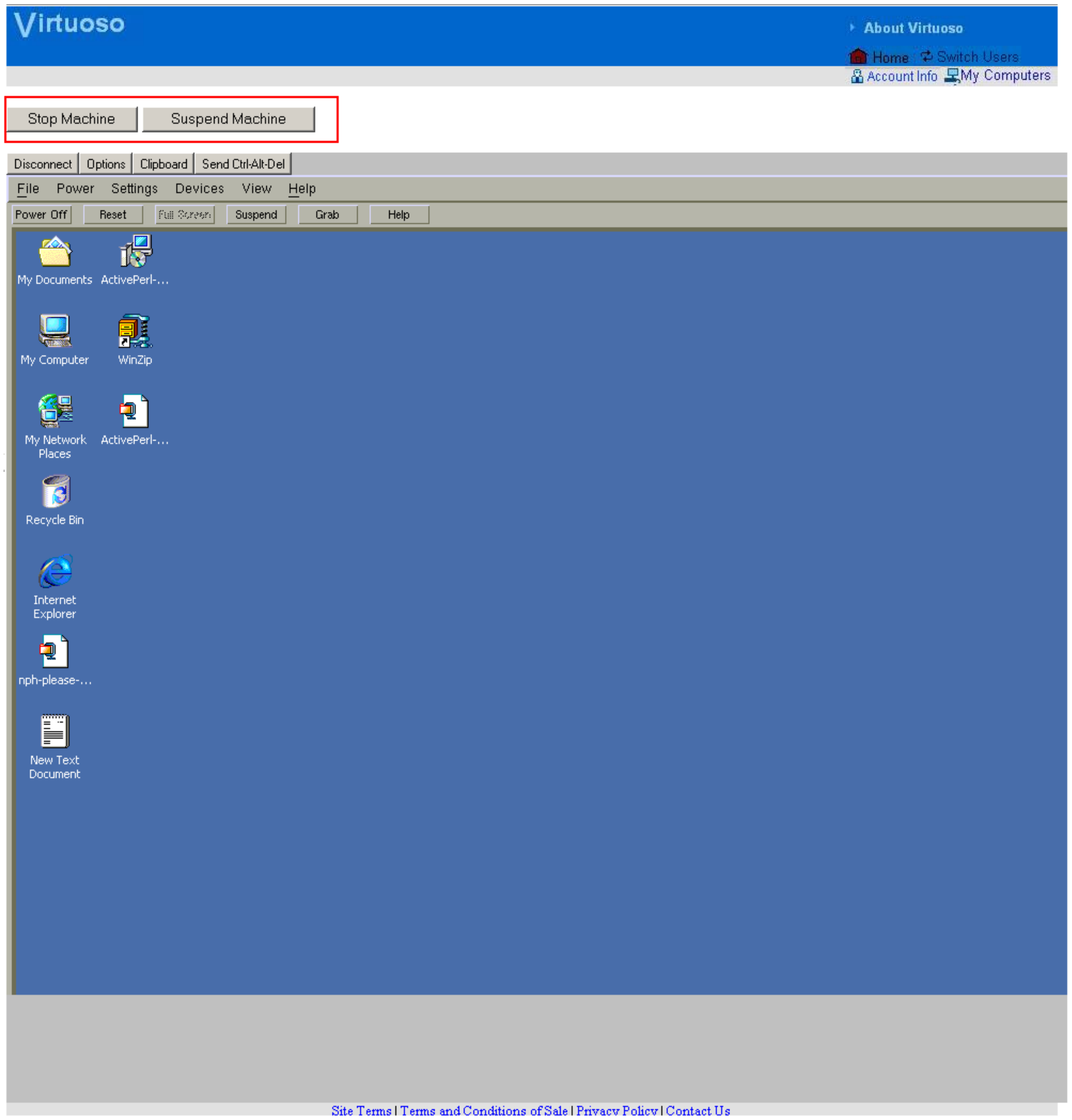
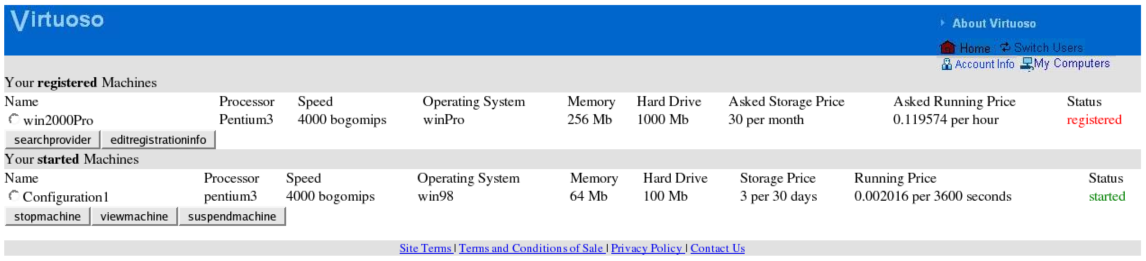


Figure 18: Buyer viewing a preconfigured machine.



Your **registered** Machines

Name	Processor	Speed	Operating System	Memory	Hard Drive	Asked Storage Price	Asked Running Price	Status
win2000Pro	Pentium3	4000 bogomips	winPro	256 Mb	1000 Mb	30 per month	0.119574 per hour	registered

searchprovider | editregistrationinfo

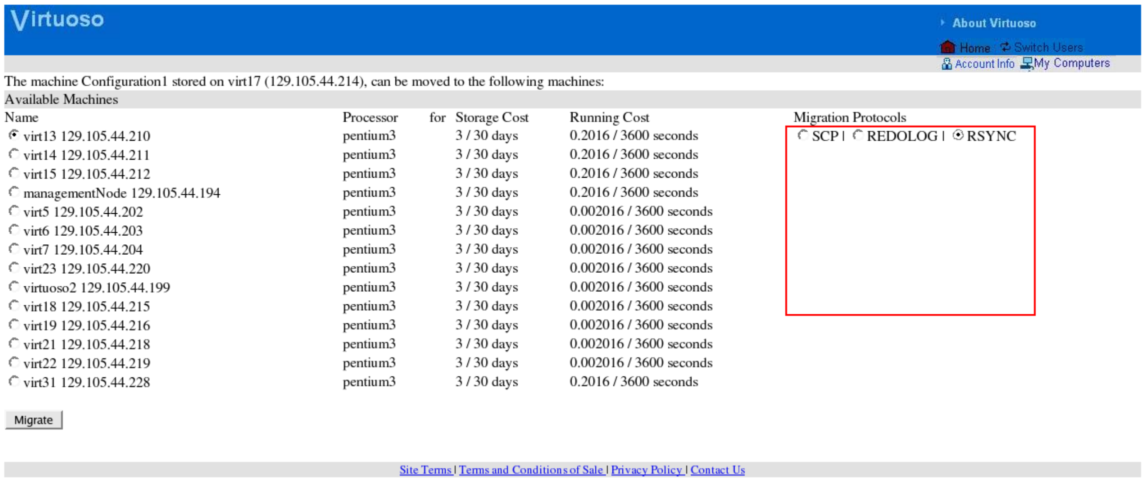
Your **started** Machines

Name	Processor	Speed	Operating System	Memory	Hard Drive	Storage Price	Running Price	Status
Configuration1	pentium3	4000 bogomips	win98	64 Mb	100 Mb	3 per 30 days	0.002016 per 3600 seconds	started

stopmachine | viewmachine | suspendmachine

Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 19: Buyer controlling machines in different states.

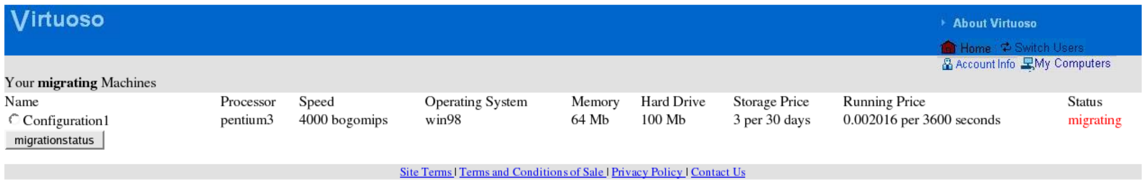

Virtuoso
About Virtuoso
Home
Switch Users
Account Info
My Computers
Migrate
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

The machine Configuration1 stored on virt17 (129.105.44.214), can be moved to the following machines:

Available Machines

Name	Processor	for Storage Cost	Running Cost	Migration Protocols
[✓ virt13 129.105.44.210]	pentium3	3 / 30 days	0.2016 / 3600 seconds	SCP | REDOLOG | [✓ RSYNC]
virt14 129.105.44.211	pentium3	3 / 30 days	0.2016 / 3600 seconds	
virt15 129.105.44.212	pentium3	3 / 30 days	0.2016 / 3600 seconds	
managementNode 129.105.44.194	pentium3	3 / 30 days	0.2016 / 3600 seconds	
virt5 129.105.44.202	pentium3	3 / 30 days	0.002016 / 3600 seconds	
virt6 129.105.44.203	pentium3	3 / 30 days	0.002016 / 3600 seconds	
virt7 129.105.44.204	pentium3	3 / 30 days	0.002016 / 3600 seconds	
virt23 129.105.44.220	pentium3	3 / 30 days	0.002016 / 3600 seconds	
virtuoso2 129.105.44.199	pentium3	3 / 30 days	0.002016 / 3600 seconds	
virt18 129.105.44.215	pentium3	3 / 30 days	0.002016 / 3600 seconds	
virt19 129.105.44.216	pentium3	3 / 30 days	0.002016 / 3600 seconds	
virt21 129.105.44.218	pentium3	3 / 30 days	0.002016 / 3600 seconds	
virt22 129.105.44.219	pentium3	3 / 30 days	0.002016 / 3600 seconds	
virt31 129.105.44.228	pentium3	3 / 30 days	0.2016 / 3600 seconds	

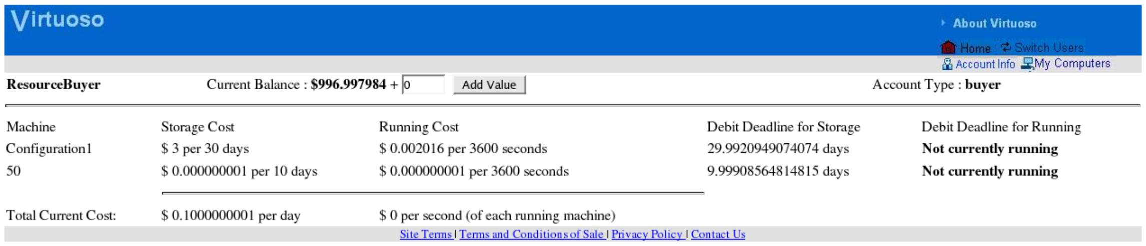
Figure 20: Buyer choosing a provider to whom they wish to migrate.



Your **migrating** Machines

Name	Processor	Speed	Operating System	Memory	Hard Drive	Storage Price	Running Price	Status
Configuration1	pentium3	4000 bogomips	win98	64 Mb	100 Mb	3 per 30 days	0.002016 per 3600 seconds	migrating

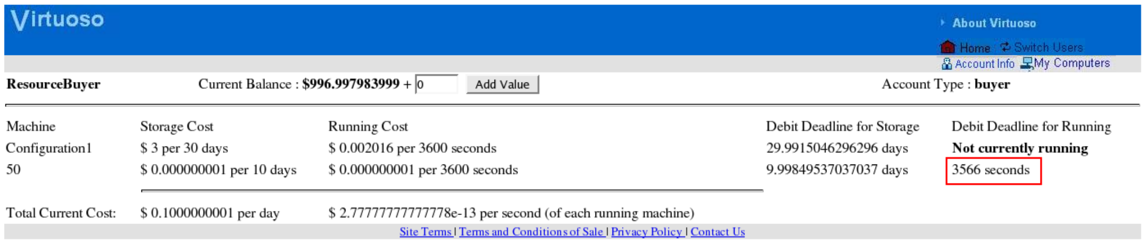
Figure 21: The machine is migrating.


Virtuoso
About Virtuoso
Home
Switch Users
Account Info
My Computers
Add Value
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

ResourceBuyer Current Balance : **$996.997984** + 0 Account Type : **buyer**

Machine	Storage Cost	Running Cost	Debit Deadline for Storage	Debit Deadline for Running
Configuration1	$ 3 per 30 days	$ 0.002016 per 3600 seconds	29.9920949074074 days	**Not currently running**
50	$ 0.000000001 per 10 days	$ 0.000000001 per 3600 seconds	9.99908564814815 days	**Not currently running**
Total Current Cost:	$ 0.1000000001 per day	$ 0 per second (of each running machine)		

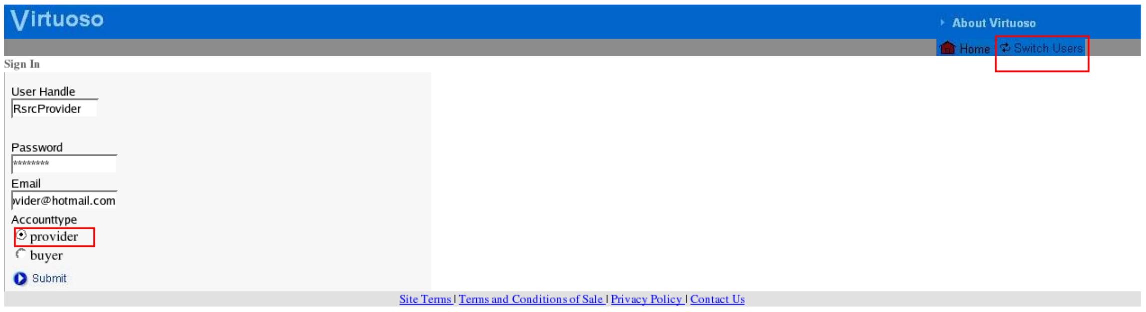
Figure 22: Buyer viewing the costs of each of his machines.


Virtuoso
About Virtuoso
Home
Switch Users
Account Info
My Computers
Add Value
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

ResourceBuyer Current Balance : **$996.997983999** + 0 Account Type : **buyer**

Machine	Storage Cost	Running Cost	Debit Deadline for Storage	Debit Deadline for Running
Configuration1	$ 3 per 30 days	$ 0.002016 per 3600 seconds	29.9915046296296 days	**Not currently running**
50	$ 0.000000001 per 10 days	$ 0.000000001 per 3600 seconds	9.99849537037037 days	3566 seconds
Total Current Cost:	$ 0.1000000001 per day	$ 2.77777777777778e-13 per second (of each running machine)		

Figure 23: Machines are only debited for their memory and speed costs while running.



Sign In

User Handle
RsrcProvider

Password
****

Email
vider@hotmail.com

Accounttype
[✓ provider]
buyer

Submit

Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 24: Provider signing up for an account.

ure 23 is a sample user account. The system allows the user to stop the machine and will wait for the machine to be started again before continuing to charge for running costs—that is, Virtuoso does not round off a time slice when it charges. The account balances are updated by daemons running on the front-end machine.

The charge to the account happens at the beginning of the billing cycle. If this were not the case, then the user could wait until their time had almost run out and then migrate the machine. One interesting result of this is that the user is slightly discouraged from migrating the machine often—this is because they will be charged for storage as soon as the machine gets to its destination.

5.2 Provider example

The experience for a provider is different from that of a buyer. In the current implementation of the system, a provider is more passive than a buyer; they do not have the option to search for buyers. The functions that they are able to perform are to add a physical machine to the system and then to control its pricing and resource options to try to attract buyers.

5.2.1 User registration

To sign up for a provider account, the user goes through a similar procedure as when signing up for a buyer account. Figure 24 shows that the user account **RscrProvider** will now be a provider account.

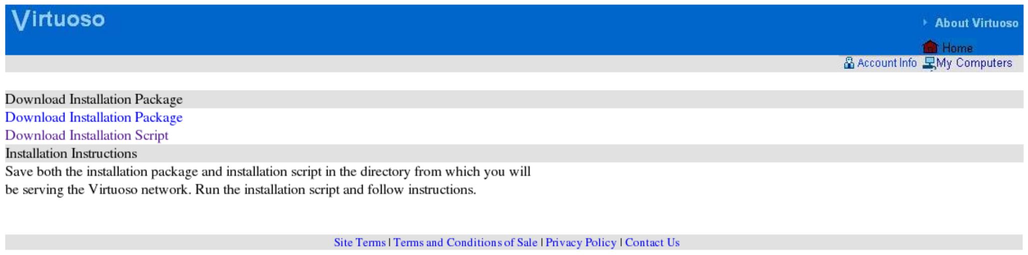
5.2.2 Adding a physical machine

When a provider first signs up, he has no machines yet and so cannot profit from the system. The provider will want to begin adding machines to the system so that buyers begin to use their resources. For a provider, the first step in adding a machine to the system is to install the provider system on the machine which they will add to the system.

Figure 25 shows the installation page for the provider system. The provider will want to download both the installation script and the gzipped file containing the provider system. See Section 3.5 for a full description of how to install the provider system.

After installing the provider system on a machine and starting the system, the user can then add that machine to the Virtuoso marketplace. Figure 26 shows the **my computers** screen for a provider—since no

because it has run out of memory and speed with respect to the Virtuoso system, while at the same time there are **no** machines in the **running** state at the time.


Virtuoso
About Virtuoso
Home
Account Info
My Computers
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Download Installation Package

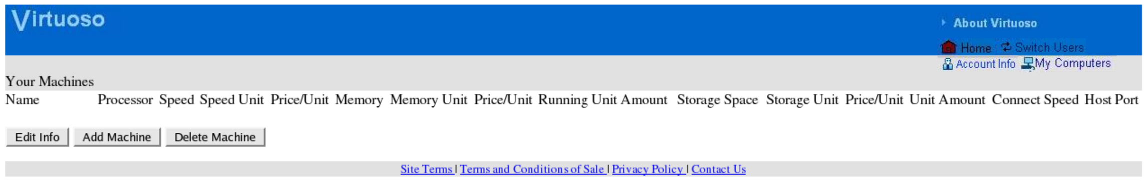
Download Installation Package

Download Installation Script

Installation Instructions

Save both the installation package and installation script in the directory from which you will be serving the Virtuoso network. Run the installation script and follow instructions.

Figure 25: Provider has no machines yet.



Your Machines

Name	Processor	Speed	Speed Unit	Price/Unit	Memory	Memory Unit	Price/Unit	Running Unit Amount	Storage Space	Storage Unit	Price/Unit	Unit Amount	Connect Speed	Host Port

Figure 26: Provider has no machines yet.

machines have been added yet, this screen is empty.

Figure 27 shows the result of choosing the **add machine** option. The provider is asked to fill in the entire description of their physical machine along with the asked price for the machine's resources. Once a machine is registered, many of its attributes' values are locked and the user will not have the ability to change them.[16]

The **port** field in the form is the port used for communication between the front-end and provider systems will be sent—this is the same port that the user specified when they installed the provider package on their machine.

Figure 28 shows the result of adding a machine to the system.

5.2.3 Editing a machine

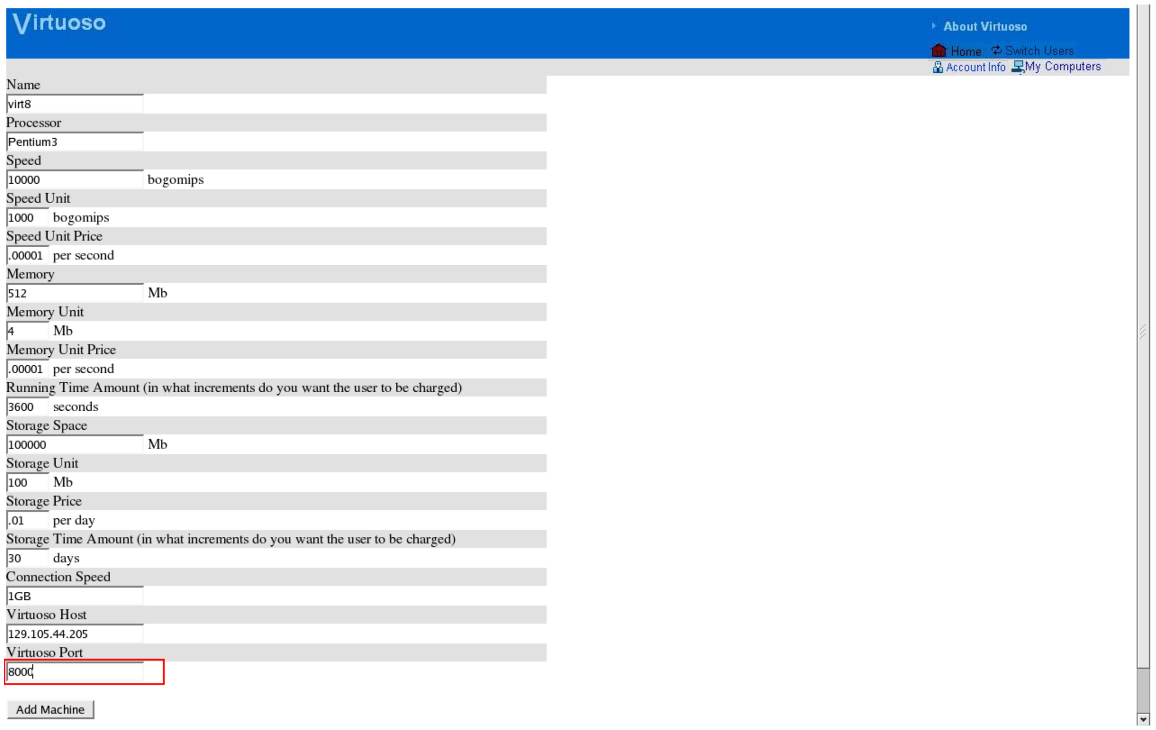
After a physical machine is added to the system, the provider has the option of editing certain aspects of the machine. They can change the service rate price for each resource and the chunks (the speed, memory, and storage "units") in which they sell those resources. These changes do not occur retroactively. That is, if a buyer has already registered a virtual machine on their physical machine, an increase or a decrease in price will not affect how much the user is charged. Figure 29 shows that the provider is prevented from changing certain fields in their machine's registration—once the machine has been added these fields are locked.

5.2.4 Account management

One of the main reasons that a provider would want to use the Virtuoso system is to recoup some of the costs incurred in buying physical resources, or to purchase resources explicitly for the purpose of earning a profit. The **my accounts** function allows the user to view their income from each physical machine in the system and also to view a break down of how much each virtual machine on that physical machine is being charged.

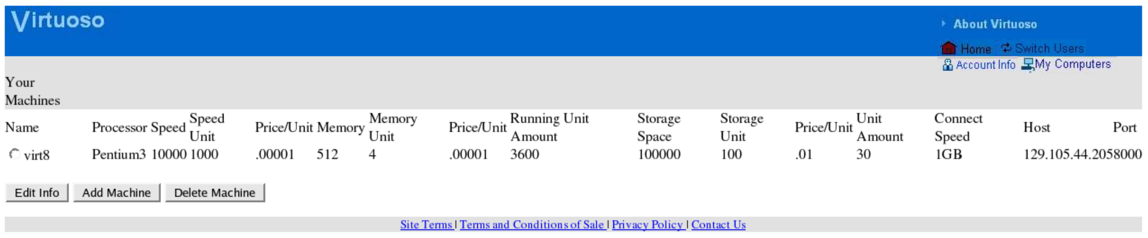
Figure 30 shows the user's account before anyone has stored a virtual machine on one of his machines.

[16]Namely, the user will not be able to edit the total amount of **storage** , **memory** , and **speed** that the machine has.



Name	virt8
Processor	Pentium3
Speed	10000 bogomips
Speed Unit	1000 bogomips
Speed Unit Price	.00001 per second
Memory	512 Mb
Memory Unit	4 Mb
Memory Unit Price	.00001 per second
Running Time Amount (in what increments do you want the user to be charged)	3600 seconds
Storage Space	100000 Mb
Storage Unit	100 Mb
Storage Price	.01 per day
Storage Time Amount (in what increments do you want the user to be charged)	30 days
Connection Speed	1GB
Virtuoso Host	129.105.44.205
Virtuoso Port	8000

Figure 27: Provider adding a new machine.



Your Machines

Name	Processor	Speed	Speed Unit	Price/Unit	Memory	Memory Unit	Price/Unit	Running Unit Amount	Storage Space	Storage Unit	Price/Unit	Unit Amount	Connect Speed	Host	Port
virt8	Pentium3	10000	1000	.00001	512	4	.00001	3600	100000	100	.01	30	1GB	129.105.44.205	8000

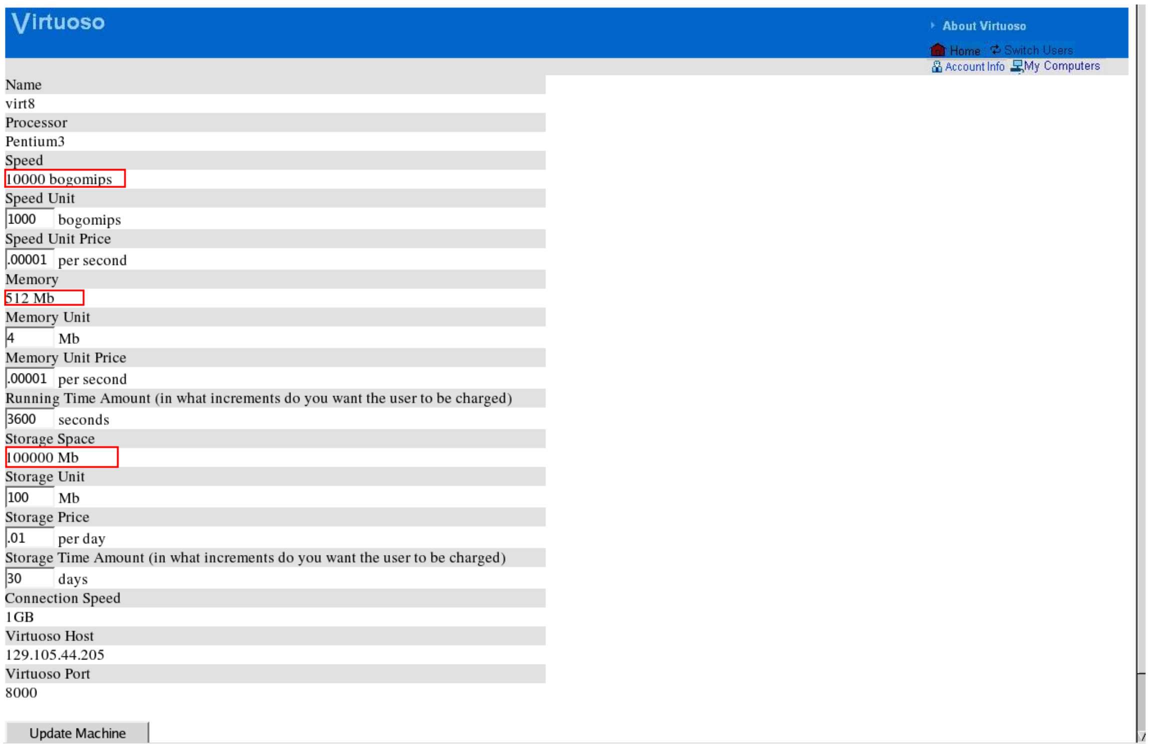
Figure 28: Provider viewing added machine.

In Figure 31, a buyer is searching for a provider on which to store a new machine of his. In this case, the buyer is choosing a machine which belongs to **RsrcProvider** and will be charged the amount that the provider is asking for that machine.

Figure 31 shows an interesting result of splitting the cost of the machine into storage and into running costs. The user has chosen a machine which is very cheap to store but very expensive to run. This type of decision will be made by buyers who know the major function for which the machine will be used. If the machine will store massive amounts of data but will not be used often, the buyer will search for providers that can fulfill that requirement at the best value. The same is true if the buyer intends to run the machine heavily but not use much storage space. If a provider knows this fact, they can take advantage of it to try to cater to one type of user or the other.[17]

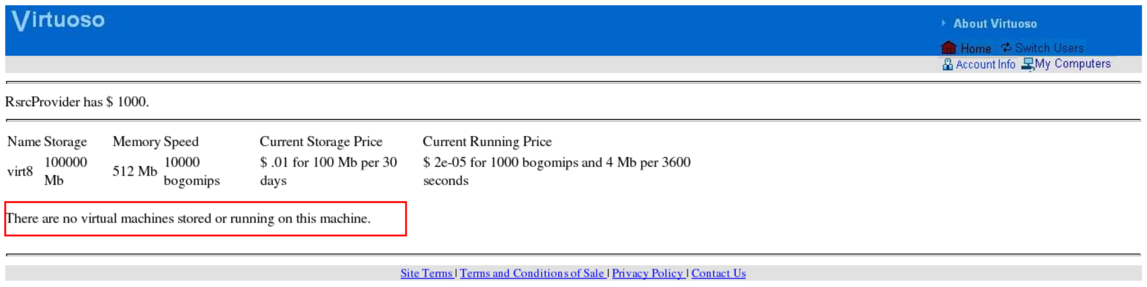
Once a buyer has stored a virtual machine on the provider's physical machine, the provider will begin to

[17]There is also a third and fourth type of user. The third type is a user who wants a good value for storage and also a good value for running of the machine. The fourth type of user doesn't know what he wants.



Name	virt8
Processor	Pentium3
Speed	10000 bogomips
Speed Unit	1000 bogomips
Speed Unit Price	.00001 per second
Memory	512 Mb
Memory Unit	4 Mb
Memory Unit Price	.00001 per second
Running Time Amount (in what increments do you want the user to be charged)	3600 seconds
Storage Space	100000 Mb
Storage Unit	100 Mb
Storage Price	.01 per day
Storage Time Amount (in what increments do you want the user to be charged)	30 days
Connection Speed	1GB
Virtuoso Host	129.105.44.205
Virtuoso Port	8000

Figure 29: Provider editing machine settings.


Virtuoso
About Virtuoso
Home
Switch Users
Account Info
My Computers
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

RsrcProvider has $ 1000.

Name	Storage	Memory	Speed	Current Storage Price	Current Running Price
virt8	100000 Mb	512 Mb	10000 bogomips	$ .01 for 100 Mb per 30 days	$ 2e-05 for 1000 bogomips and 4 Mb per 3600 seconds

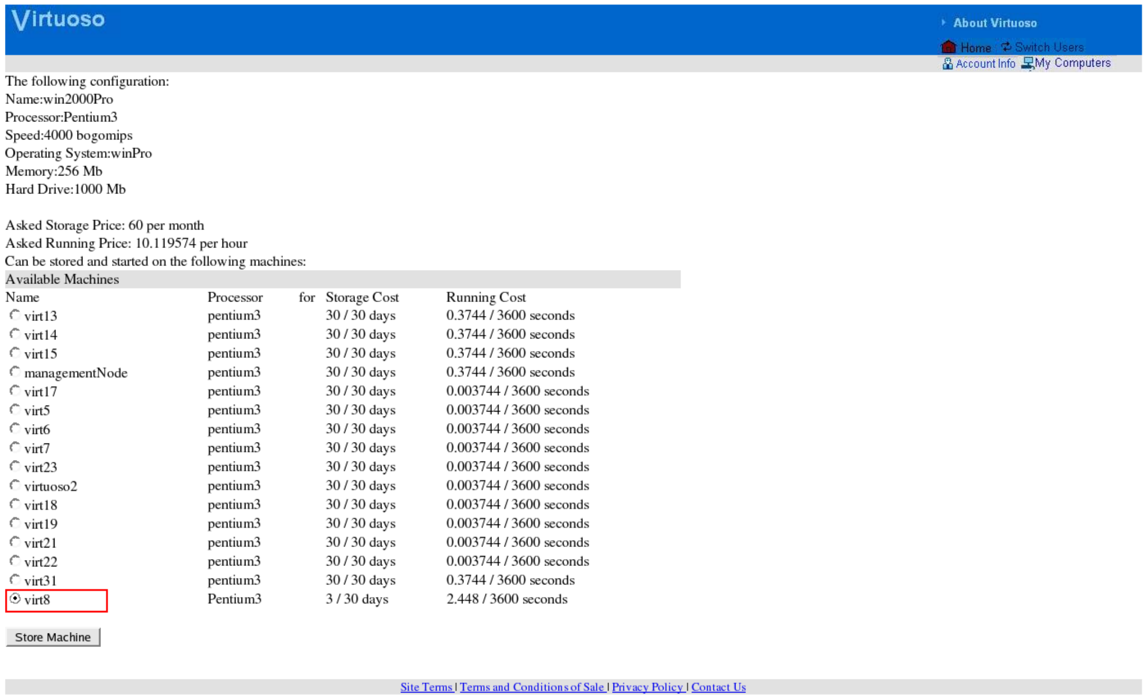
There are no virtual machines stored or running on this machine.

Figure 30: Provider viewing his account before any buyers are using his machines.

receive deposits to his account. The storage deposit will be made at intervals that the provider specified. A buyer is charged the running cost of a machine only when the machine is in the **running** state.

5.2.5 Viewing the provider marketplace

A buyer has the option of seeing an average market price for standard configurations of virtual machines—this is so that they can have a rough guide as to how much to bid for the resources they want. Similarly, a provider has the option to view the market averages for physical resources so that they know approximately how much to charge. This is provided by the **provider marketplace** function on the home page. Figure 33 shows a typical display of all of the providers in the Virtuoso system.



The following configuration:
Name:win2000Pro
Processor:Pentium3
Speed:4000 bogomips
Operating System:winPro
Memory:256 Mb
Hard Drive:1000 Mb

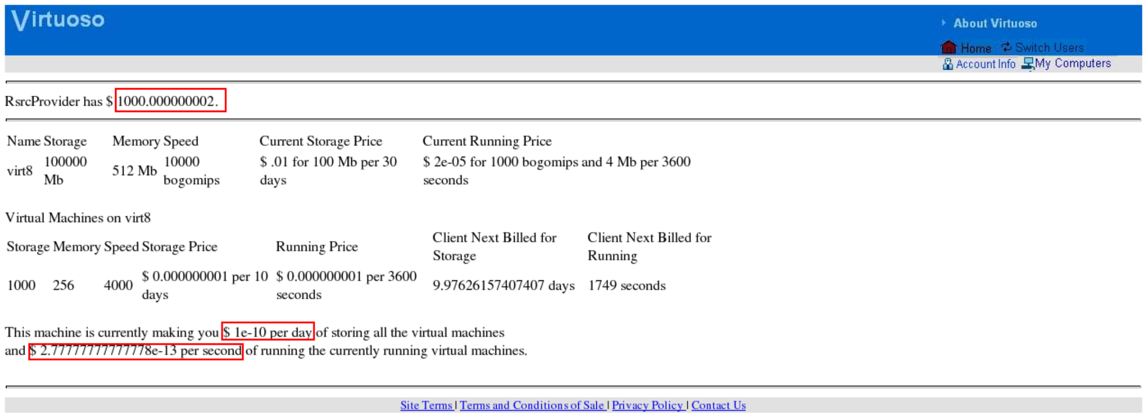
Asked Storage Price: 60 per month
Asked Running Price: 10.119574 per hour
Can be stored and started on the following machines:

Available Machines

Name	Processor	for Storage Cost	Running Cost
virt13	pentium3	30 / 30 days	0.3744 / 3600 seconds
virt14	pentium3	30 / 30 days	0.3744 / 3600 seconds
virt15	pentium3	30 / 30 days	0.3744 / 3600 seconds
managementNode	pentium3	30 / 30 days	0.3744 / 3600 seconds
virt17	pentium3	30 / 30 days	0.003744 / 3600 seconds
virt5	pentium3	30 / 30 days	0.003744 / 3600 seconds
virt6	pentium3	30 / 30 days	0.003744 / 3600 seconds
virt7	pentium3	30 / 30 days	0.003744 / 3600 seconds
virt23	pentium3	30 / 30 days	0.003744 / 3600 seconds
virtuoso2	pentium3	30 / 30 days	0.003744 / 3600 seconds
virt18	pentium3	30 / 30 days	0.003744 / 3600 seconds
virt19	pentium3	30 / 30 days	0.003744 / 3600 seconds
virt21	pentium3	30 / 30 days	0.003744 / 3600 seconds
virt22	pentium3	30 / 30 days	0.003744 / 3600 seconds
virt31	pentium3	30 / 30 days	0.3744 / 3600 seconds
[✓ virt8]	Pentium3	3 / 30 days	2.448 / 3600 seconds

Store Machine
Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 31: Buyer stores a machine on one of the provider's machines.



RsrcProvider has $ 1000.000000002.

Name	Storage	Memory	Speed	Current Storage Price	Current Running Price
virt8	100000 Mb	512 Mb	10000 bogomips	$ .01 for 100 Mb per 30 days	$ 2e-05 for 1000 bogomips and 4 Mb per 3600 seconds

Virtual Machines on virt8

Storage	Memory	Speed	Storage Price	Running Price	Client Next Billed for Storage	Client Next Billed for Running
1000	256	4000	$ 0.000000001 per 10 days	$ 0.000000001 per 3600 seconds	9.97626157407407 days	1749 seconds

This machine is currently making you $ 1e-10 per day of storing all the virtual machines
and $ 2.77777777777778e-13 per second of running the currently running virtual machines.

Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

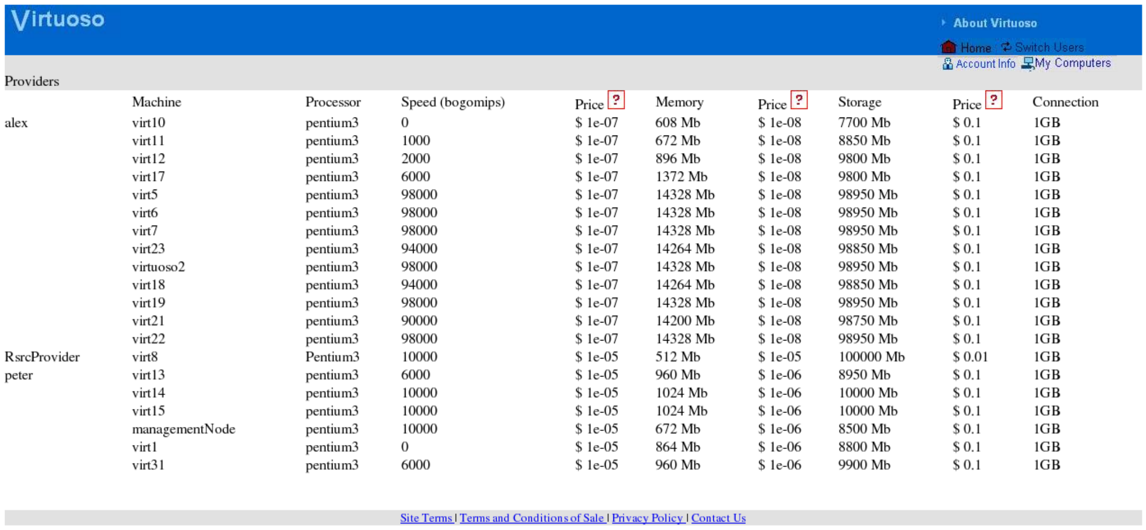
Figure 32: Provider viewing physical machines and virtual machines stored on them.

6 Implementation

Here we give a thorough description of the Virtuoso implementation, except for the migration system, which is described in the next section.

6.1 Front-end system

The front-end of Virtuoso is what the users interact with. It is composed of five integral components and the event logger. The sections are the **web interface**, the **database**, the **market daemons**, the **machine control interface**, and the **migration control interface**.



Providers

	Machine	Processor	Speed (bogomips)	Price ?	Memory	Price ?	Storage	Price ?	Connection
alex	virt10	pentium3	0	$ 1e-07	608 Mb	$ 1e-08	7700 Mb	$ 0.1	1GB
	virt11	pentium3	1000	$ 1e-07	672 Mb	$ 1e-08	8850 Mb	$ 0.1	1GB
	virt12	pentium3	2000	$ 1e-07	896 Mb	$ 1e-08	9800 Mb	$ 0.1	1GB
	virt17	pentium3	6000	$ 1e-07	1372 Mb	$ 1e-08	9800 Mb	$ 0.1	1GB
	virt5	pentium3	98000	$ 1e-07	14328 Mb	$ 1e-08	98950 Mb	$ 0.1	1GB
	virt6	pentium3	98000	$ 1e-07	14328 Mb	$ 1e-08	98950 Mb	$ 0.1	1GB
	virt7	pentium3	98000	$ 1e-07	14328 Mb	$ 1e-08	98950 Mb	$ 0.1	1GB
	virt23	pentium3	94000	$ 1e-07	14264 Mb	$ 1e-08	98850 Mb	$ 0.1	1GB
	virtuoso2	pentium3	98000	$ 1e-07	14328 Mb	$ 1e-08	98950 Mb	$ 0.1	1GB
	virt18	pentium3	94000	$ 1e-07	14264 Mb	$ 1e-08	98850 Mb	$ 0.1	1GB
	virt19	pentium3	98000	$ 1e-07	14328 Mb	$ 1e-08	98950 Mb	$ 0.1	1GB
	virt21	pentium3	90000	$ 1e-07	14200 Mb	$ 1e-08	98750 Mb	$ 0.1	1GB
	virt22	pentium3	98000	$ 1e-07	14328 Mb	$ 1e-08	98950 Mb	$ 0.1	1GB
RsrcProvider	virt8	Pentium3	10000	$ 1e-05	512 Mb	$ 1e-05	100000 Mb	$ 0.01	1GB
peter	virt13	pentium3	6000	$ 1e-05	960 Mb	$ 1e-06	8950 Mb	$ 0.1	1GB
	virt14	pentium3	10000	$ 1e-05	1024 Mb	$ 1e-06	10000 Mb	$ 0.1	1GB
	virt15	pentium3	10000	$ 1e-05	1024 Mb	$ 1e-06	10000 Mb	$ 0.1	1GB
	managementNode	pentium3	10000	$ 1e-05	672 Mb	$ 1e-06	8500 Mb	$ 0.1	1GB
	virt1	pentium3	0	$ 1e-05	864 Mb	$ 1e-06	8800 Mb	$ 0.1	1GB
	virt31	pentium3	6000	$ 1e-05	960 Mb	$ 1e-06	9900 Mb	$ 0.1	1GB

Site Terms | Terms and Conditions of Sale | Privacy Policy | Contact Us

Figure 33: Viewing provider marketplace.

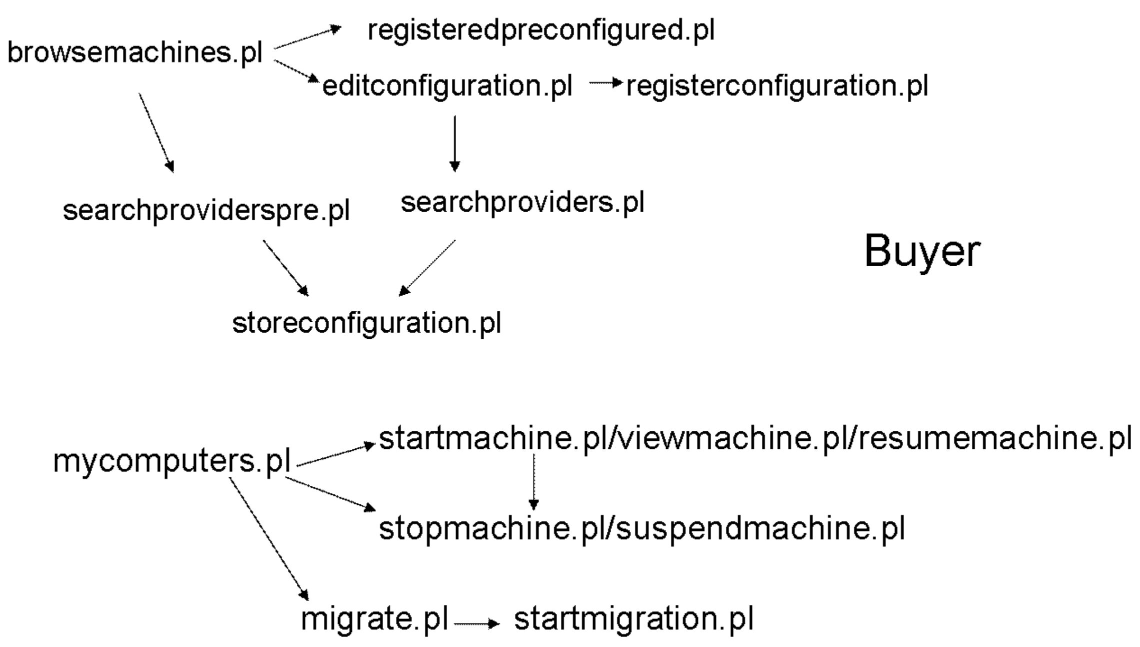


Figure 34: Links between scripts for a buyer.

6.1.1 Web interface

The interface that both the buyer and provider users access the system through is a set of Perl scripts which use Perl's CGI package. Most of the scripts are forms which submit information to the Virtuoso system through method POST. The links between the scripts for a buyer are shown in Figure 34, and for a provider in Figure 35.

The scripts that comprise the interface are:

signup.pl user signs up for an account.

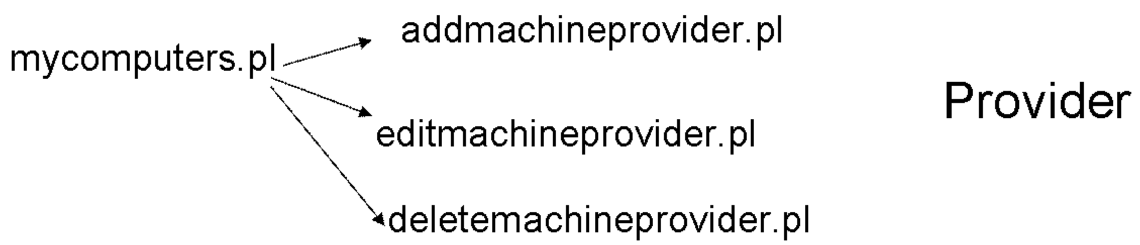


Figure 35: Links between scripts for a provider.

login.pl user logs in to account.

switchlogin.pl logs the users out of the system by deleting the stored cookie and then prompts the user to log in again.

browsemachines.pl user views standard virtual machine configurations in the system.

editconfiguration.pl buyer user can edit a configuration before registering.

registerconfiguration.pl gets a form from **editconfiguration.pl** and stores a registration in the database.

registerpreconfigured.pl gets a form from **browsemachines.pl** and stores a registration in the database.

mycomputers.pl displays a user's machines to the user. This script takes a different action depending on whether the accessor is a provider or a buyer.

searchproviderspre.pl gets a form from **browsemachines.pl** and searches for providers based on that yet-unregistered configuration.

searchproviders.pl gets a form from **mycomputers.pl** and searches for providers based on that registration.

storeconfiguration.pl gets a form from **searchproviders.pl** or **searchproviderspre.pl** and stores the given configuration on the chosen provider machine.

startmachine.pl gets a form from **mycomputers.pl** and then starts the given machine and then connects a vnc server and viewer; see Section 6.1.8 for more details.

viewmachine.pl gets a form from **mycomputers.pl** and then connects a vnc server and viewer; see Section 6.1.8 for more details.

stopmachine.pl gets a form from **mycomputers.pl** or **resume/start/view machine.pl** and stops the running of the given **machineid**.

suspendmachine.pl gets a form from **mycomputers.pl** or **resume/start/view machine.pl** and suspends the running of the given **machineid**.

resumemachine.pl gets a form from **mycomputers.pl** and resumes the given machine and then connects a vnc server and viewer; see Section 6.1.8 for more details.

migrate.pl gets a form from **mycomputers.pl** and searches for providers to whom the given **machineid** machine can be migrated. For each suitable host, the script then queries those hosts for the migration protocols that they support.

startmigration.pl gets a form from **migrate.pl** and initiates the migration of the given machine to the given destination.

addmachineprovider.pl gets a form from **mycomputers.pl**. If the form contains the CGI parameter **machineid**, then it searches the database and fills in a new form with a copy of that physical machine's configuration.[18] If no **machineid** is POSTed then a blank form is printed.

editmachineprovider.pl gets a form from **mycomputers.pl** and prints a form letting a provider edit the registration information of a physical machine.

deletemachineprovider.pl gets a form from **mycomputers.pl** and deletes the entry from the database. It also deletes the entries of the virtual machines stored on that machine and sends emails to their owners telling them how to get in touch with the provider.[19]

myaccounts.pl displays a user's machines and their current costs and running states to the user, along with their account balance. This script takes a different action depending on whether the accessor is a provider or a buyer.

getproviders.pl a user can view the entire marketplace of providers.

6.1.2 HTML output

The package **standard_html.pm** is used to hold standard html formats that are used by Virtuoso; namely the title bar and the footer as well as the login screen.

Other HTML output by the system is specific to each script. Each script has a printMainForm function that takes as input the items obtained from the database and from the form input to render as HTML output to the user.

6.1.3 User authentication

A user is not allowed to access any part of Virtuoso, except the homepage, without logging in. When a user logs in, a cookie is placed on their computer with a unique session ID, which is checked against the session ID stored in the database. The id is composed of the username+`rand()`.

Once the username and session id are checked against those stored in the database and are found to match, then the system will attempt to perform the action that the current script implements. However, the system will first check that action against permissible actions for the given user.

6.1.4 Access authorization

Each script is designed for a specific action or set of actions. If a user attempts to fool the script into doing an action that the user should not be permitted to do, the script should not permit that action. The system allows only those actions explicitly permitted for the user in a given context.

Illegal actions that we are especially careful to stop include:

- a buyer trying to access a machine that they do not own (i.e., passing in a form where they have faked the machineid).

[18] This is to expedite the adding of new physical machines; a provider can simply copy the configuration this way.

[19] A better way to let providers delete machines should be the subject of further development; this falls under the category of a physical machine going offline.

- a buyer trying to change the state of a machine which is not accessible from the current state (for example: calling the startmachine.pl script while the machine is in the migrating state).
- a provider attempting to access buyer pages and vice versa (for example: provider attempting to store a configuration).

6.1.5 Database access

The database access method that Virtuoso uses depends on the type of database that is specified. It is possible to switch between the internal text database and the Oracle databases by using the **switchdb.pl** tool. This tool switches the initialization of the environment variables at the beginning of each script in **public_html**. It also copies the contents of the previous database into the one that has been switched to—this way, the databases are consistent. If the database type in the **env->INIT** function call is set to ORACLE then the **oracle_db.pm** package is used, if it is set to TEXT then the **text_db.pm** package is used. These packages support the same functions and API's but obviously implement them differently.

Creating a new database structure involves calling its InitDB function, which takes a user name and password as well as the table name which will be the focus of this database structure. See Section 6.1.6 for a full description of the tables in the Virtuoso system.

The following functions are implemented:

new creates a new database structure which holds the user name, password, and particular table name.

InitDB initializes the database structure's member variables.

findRow given a key type and value and a set of rows, searches for the given key in the rows.

findUniqueId searches the database table to find a unique id. This is used when creating a new machine and the system needs to assign it a universal unique identifier (uuid). The identifier is a 128 bit number.[20]

getParam takes as input a key and a row in the database and returns the value associated with that key.

getContents returns all the rows of a table in the database in an array.

addToDb takes as input a primary key and value and a row. It adds the row to the database and overwrites a row in the database if it already exists.

deleteFromDb takes as input a primary key and value. Deletes the row associated with the key if it exists.

updateKeyDb takes as input a primary key and value as well as a secondary key and value. It finds the row in the table associated with the given primary key and sets the value of the secondary key to the new value.

updateKeyChangeDb takes as input a primary key and value as well as a secondary key and delta value. It finds the row in the table associated with the given primary key and sets the value of the secondary key to be the $oldvalue - deltavalue$. [21]

shut this call frees all the member variables in the database structure. [22]

[20]a.k.a. global unique identifier (gid)

[21]In a couple of scripts you will see the function call updateKeyChangeDb{primarykey,value,secondarykey, -deltavalue}. Passing in a negative value will increase the old value in the database by the delta value instead of decreasing it.

[22]Since we're working in Perl, this is rather silly and isn't used.

6.1.6 Database tables

When the front-end is first installed, it creates a set of standard tables on the front-end machine—each field in the table is a character string of size 50. These tables are:

- Tables **config and preconfig** are tables of all the standard configurations stored in the system—preconfig holds those configurations which already have an operating system installed. Its fields are:

  machineid primary key, gets replaced by uuid when machine is registered.

  askedstorageprice market average of the price for the given storage amount.

  askedrunningprice market average of the price for the given speed and memory amount

  configname a human-readable name.

  processor the processor type.

  os the operating system type.

  memory the amount of memory.

  hd the amount of disk storage.

- Table **vm** contains all the virtual machines in the system. Its fields are:

  machineid universal unique identifier.

  speed number of bogomips[23] the machine has.

  storageprice total storage price that owner will be charged every **storage amount** days.

  storageamount how many days to wait between charging for storage.

  runningprice total running price that owner will be charged every **runningprice** seconds.

  runningunitamount how many seconds to wait between charging for running.

  askedstorageprice while the machine is registered, this stores the bid price that the user has entered for storage.

  askedrunningprice while the machine is registered, this stores the bid price that the user has entered for running.

  vmstatus the current state of the machine. The possible states are stored in the table **statusrelations**.

  owner the owner of the current machine.

  lastdebitedrunning the last time that the owner was debited for running.

  lastdebitedstorage the last time that the owner was debited for storage.

  version the current version of the machine (gets increased at every state change).

  configname a human-readable name.

  processor processor type.

  os operating system type.

  memory amount of memory.

  hd amount of disk storage.

[23]This is the performance of the machine. Currently, it is defined to be the Linux bogomips test.

- Table **statusrelations** lists all the possible states of a VM and the possible actions that can be performed in each. Its fields are:

  status a state of a machine.

  relations legal actions that can be performed on that state.

- Table **pmseller** lists all of the physical machines in the system. Its fields are:

  machineid universal unique identifier.

  machinename a human readable name.

  processor the processor type.

  speed number of bogomips the machine has.

  speedunit the size of the chunk of speed in which the resource will be sold.

  speedprice the price per second for the **speedunit** size chunk of speed.

  memory number of megabytes of memory the machine has.

  memoryunit the size of the chunk of memory in which the resource will be sold.

  memoryprice the price per second for the **memoryunit** size chunk of memory.

  runningunitamount the period (in seconds) with which a user storing their VM on this machine will be charged for running.

  storage number of megabytes of storage space the machine has.

  storageunit the size of the chunk of storage in which the resource will be sold.

  storageprice the price per day for the **storageunit** size chunk of storage.

  storageamount the period (in days) with which a user storing their VM on this machine will be charged for storage.

  connectspeed speed of network (for example: 1Gbps).

  virtuosohost ip address of the machine.

  virtuosoport port on which **virtuoso.pl** is listening.

  owner the owner of the current machine.

- Table **machinepairings** associates physical machines with the virtual machines they contain. Its fields are:

  machineid universal unique identifier of a virtual machine.

  pmachineid universal unique identifier of a physical machine.

- Table **users** lists all the users in the system. Its fields are:

  username user login name

  password must be atleast 8 characters (vncpasswd requires this)

  sessionID username + rand()

  emailaddress contact info

  balance currency in the system that the user holds (dollars)

  accounttype either buyer or provider

6.1.7 Virtual machine control

The control of the state of the virtual machine is defined by the **statusrelations** table. For a given state, there are only a certain set of actions that can be taken. This table is initialized when the system is first installed.

The states and actions that may be performed are:

migrating migrationstatus

registered searchprovider editregistrationinfo

started stopmachine viewmachine suspendmachine

stopped startmachine migrate

stored startmachine migrate

suspended resumemachine migrate

> The fact that each relation name can be turned into the name of a script by adding a ".pl" is not coincidental. This is done in a Javascript function printed out by the **mycomputers.pl** interface. That is, for a given state, a set of Javascript functions will be generated which are referenced by buttons in the interface and which call the respective script to run.

6.1.8 VNC display

Virtuoso uses a Java applet VNC client to let a buyer view his running machines. When a buyer asks to view a machine, a message is passed to the provider computer asking to first disconnect the machine from a VNC session—if the machine is not in a VNC session then this message is ignored. After the success of the disconnect, a connect message is sent to the provider machine which returns a port on which the VNC server will allow connections to the machine's display. The **viewmachine.pl** and **startmachine.pl** scripts get served a Java VNC applet from the front-end machine, the applet takes as input the host machine on which the virtual machine is running and the port on which the VNC server for that session is listening.[24]

6.1.9 Background daemons

Several background daemons are run to keep the Virtuoso system up to date on marketplace trends and also to update the balance of user accounts.

The script **priceaverage.pl** runs through the configurations and preconfigurations stored in the database tables **config** and **preconfig** and updates the average market prices for each configuration. For each configuration it searches for providers that would be able to host the configuration and then, from the suitable providers, it makes an average for the resource prices. If it is not able to find a provider which could host the configuration, it leaves the stale price. This script is run, by default, every 60 seconds.

The scripts **storageupdate.pl** and **runningupdate.pl** update the balances of both the provider and the buyer for a given machine if that machine's storage deadline or running deadline has been reached. The provider is credited by the storageprice/running price if the storage/running deadline has been reached. The system can figure out whether a storage/running deadline has been reached by seeing whether the current time $\geq$ the last time debited + storage or running unit amount. Machines are only debited for CPU when they are in the **running** state.

[24]Currently there is no mechanism to dynamically adjust the VNC display size and it is set to be large enough to accommodate a typical display screen.

6.1.10 Event logger

The event logger is a debugging tool for the Virtuoso system and is helpful in understanding how the system works. It listens on port 6767 for connections and understands the messages **append_to_log** and **flush_log**. The logger does not actually flush to a file until the **flush_log** command is received. All important events in the system (including provider system events) are logged through the event logger and come in order of their occurrence. Thus, to read an event log is to see the order of Virtuoso calls and the origin of each call.[25]

6.1.11 Environment initialization

The **env.pm** package in the **virtuosoDB** directory is called at the beginning of each script to initialize key environment variables. During installation, the user's environment was read and printed to the **env.pm** file. This is important since the CGI scripts are executed by the user **oracleuser** and not by the user **virtuoso**, so the environment would be different unless explicitly set to the same thing.

6.2 Provider system

The provider system is mostly a set of Perl scripts and packages which work together to implement the functions of controlling virtual machines, answering questions about those machines during migration events, answering VMMP (migration) messages, and migrating machines. Since a full description of the migration system is available in Section 7, we focus on aspects of virtual machine control here.

6.2.1 Files

The following lists the files relevant to the provider system. **prog_dir** refers to the directory in which the provider scripts and other tools are installed.

~/.vnc/xstartup VNC startup script. It must be configured to execute prog_dir/console_starter.pl. If it does not exist, it will be automatically created. The console starter runs the VMWare console session as the window manager.

prog_dir/virt_config.pm Contains all of the configuration parameters for the provider.

prog_dir/vm_config_create.pm Perl library that implements the function create_from_config(@configuration_parameters) that creates virtual machines based on configuration parameters supplied by Virtuoso. Translates a Virtuoso config file to a VMWare config file.

prog_dir/vm_delete.pm Perl library that provides the mechanism to delete virtual machines from the provider, delete_vm($uuid).

prog_dir/virt_interface.pm Perl library that implements the client side interface to the Virtuoso provider network interface. Implements:

- create_vm($host, $port, @configuration_parameters)
- register_vm($host, $port, $uuid)
- start_vm($host, $port,$uuid)
- stop_vm($host, $port, $uuid)

[25]If the event logger is not operating, then all events will be logged locally by the provider system. This is useful to speed up operation of the system since every event will no longer have to wait to send a log of itself to the central server.

- set_passwd($host, $port, $uuid)
- $port = connect_vm($host, $port, $uuid)
- disconnect_vm($host, $port, $uuid)
- delete_vm($host, $port, $uuid)
- @info = get_server_info($host, $port, $uuid)

prog_dir/virt_machine_info.pm Collects performance and configuration information from the machine the server is running on. The library implements @info = get_info().

prog_dir/vm_control.pm Implements the interface to the VMWare Perl APIs, providing

- stop_vm($uuid)
- start_vm($uuid)
- resume_vm($uuid)
- suspend_vm($uuid)
- register_vm($uuid)
- $status = vm_status($uuid)

prog_dir/vm_lookup.pm Implements the mapping from uuid to VMWare configuration file location. This is needed because Virtuoso tracks machines by their uuid, while VMWare and disk operations need the path to the configuration file. It provides the following:

- add_vm($uuid, $config_file)
- delete_vm($uuid)
- stash_vm($uuid)
- $config_path = find_vm($uuid)
- restore_vm($uuid)

prog_dir/vm_ports.pm This implements the mapping of a virtual machine to the port number that it runs VNC on. It also tracks how many VNC sessions are running, and limits the number of simultaneous sessions that can be run. It provides:

- add_vm($uuid)
- delete_vm($uuid)
- find_vm($uuid)

prog_dir/vnc_control.pm This implements the controls for the VNC software, that provides the interface to the Virtuoso users. It implements:

- start_vnc($uuid)
- stop_vnc($uuid)
- set_vnc_passwd($uuid, $passwd)

prog_dir/vncserver This is a modified version of the vncserver script found in the standard VNC distribution. It allows us to handle multiple sessions easily.

prog_dir/vnc_mod/* This is a modified version of the vncpasswd program that creates VNC passwords. The difference is that this version takes the password on the command line.

prog_dir/virtuoso.pl This is the actual provider daemon that is run in the background continuously. It handles all the communication with the front end.

prog_dir/console_starter.pl This is run from the VNC xstartup file. It takes the uuid as a command line argument, looks up the configuration file path, and then forks and execs vmware-console with the correct configuration file. In other words, it attaches a display to a running VM.

machine_dir refers to the directory in which VMs will be stored on the provider.

machine_dir/vm_list.virt Virtuoso data file that contains uuid to configuration file path mapping.

machine_dir/vm_ports.virt Virtuoso data file that contains the uuid to VNC port mapping.

For each VM, the following is stored in addition to the actual VMWare state:

machine_dir/vm/vm.passwd VNC password file for the VM.

machine_dir/vm/vm.virt Virtuoso configuration parameters.

machine_dir/vm/vm.pid VNC pid for stopping the VNC service for the VM.

The following sections describe the functionality of the provider server in more detail.

6.2.2 Local state

To keep track of the machines stored locally in the **virt** directory and to keep track of certain configuration information, the provider system manages a set of local state files. These files are key, value pairs and contain information about machines.

vm_list.virt pairs each machine's uuid with the relative path to the machine's configuration file.

vm_origin_list.virt pairs each machine's uuid with the IP address and port from which the machine was migrated. If the machine originated on this computer, there is no entry for that machine.

vm_version_list.virt pairs each machine's uuid with the version number stored locally. This is consistent with the version number stored in the central database.

vm_port_list.virt keeps a list of ports that can be used for VNC sessions. Pairs a port with a machine id if that machine is connected to a VNC session.

6.2.3 Virtual machine control

The act of creating/registering/starting/stopping/suspending/resuming/deleting a virtual machine is carried out with the help of the **vm_control.pm** package. The **virtuoso.pl** server gets a message and dispatches to an action based on the type of the message. There is an assumption that if the message has been received, then the action is permissible in the system. That is, there is no access checking by the provider system.[26]

[26]This is the current state. In a deployed system, the provider would not trust the front-end and the network quite so much. One way to add additional security now is to bind the provider to the loopback address, and then set up an SSH tunnel between the provider and the front end for access to that port.

To create a virtual machine image, Virtuoso does not use any VMware related API, but rather has its own methods of creating a disk, configuration file, and ram. The **create_from_config** function in **vm_config_create.pm** takes as input an array of key, value pairs. These are the values that will be inserted into a configuration file. This function creates the directory where the machine will be manufactured, creates the disk file, and registers the machine with the local state files through **vm_lookup.pm**.

Each VMWare virtual machine is issued a unique id from the Virtuoso system, instead of the standard uuid that is automatically generated by VMWare whenever the machine is run. Also issued by Virtuoso is the Ethernet address, which, like the uuid, is usually automatically generated by VMWare when the machine is run. These numbers are controlled by Virtuoso as (1) a means of tracking and managing machines across multiple providers and (2) a way to interface to the VNET virtual network service.

Currently all of the configuration is done at the time of creation, as discussed above. In order to provide security for the providers Virtuoso users are not allowed to modify the machine configurations themselves. The actual configuration file is locked to prevent this from happening. Providing for configuration changes is due in a future iteration of this distribution.

All other machine control commands then are passed through VMWare. The package **vm_control.pm** is a wrapper for the VMWare Perl API. Refer to the VMware Perl Scripting API documentation at **http://www.vmware.com/pdf/Scripting_API_21.pdf** for a full description of all supported functions. The intent behind wrapping this API is to make it possible in the future to use similar APIs provided by other VMMs.

To **start** a machine, the system needs to first find its configuration file for the given **uuid**. The **vm_lookup.pm** package is used—it consults **vm_list.virt** to find the relative path and appends it to the Virtuoso path stored in the **virt_config.pm** file. Once the system knows where the machine is, it can use a VMWare control server to send a start message to VMWare. A similar sequence of steps is used for resuming a machine, and discovering its status.

The sequence of steps that it takes to stop a running machine is a bit more involved. This is because VMWare will often ask a question before letting a user stop the machine. A typical question might be "Would you like to commit undoable disk..." Since this is an automated system, the stop function has a set of default answers that it uses to pass to VMWare—the default is to answer "yes" to everything.[27] If the stop command is successful, the version of the machine is increased by 1.

The act of suspending a machine may fail if the use asks to suspend the machine while its operating system is still loading - VMWare refuses to honor the suspend request. The suspend command attempts to fix this by asking to the suspend the machine repeatedly if the request is not successful—it stops after 100 attempts and returns a response of failure. If the suspend command is successful, the version of the machine is increased by 1.

When the system first creates a configuration, it sets the permissions on the configuration file to be 544. The register machine function changes these, temporarily, to 744. It then registers the machine and changes the permission back to 544. The version of the machine is then increased by 1.

6.2.4 VNC control

After a machine has been started, it can be viewed through a VNC session. The control of VNC is managed by the **vnc_control.pm** package. This package exposes the functions of starting and stopping VNC and setting the VNC password. The set password function of VNC takes as input a **uuid** and password. It then calls **vncpasswd** which is an command line tool (executable) that sets the VNC password to the given

[27]It would be possible to provide an exhaustive interface to VMWare that lets the user answer these questions himself, but that would cause us to be heavily dependent on VMWare.

password. A password must contain at least 6 characters. After the password is set, the system will receive a message from the front-end to connect a vncserver.

The start VNC function takes as input a **uuid**. Using the **vm_lookup.pm** package it adds a port on which the VNC session will be started and it builds a command line to pass to **vncserver**— part of which includes the name of the password file for that particular machine. It then runs **vncserver** with the command line— this is the standard **vncserver** script except that it configures the VMM's console display as the window manager. This is done by passing **xstartup** the **uuid** of the machine— **xstartup** then calls **consolestarter.pl** to start a VMWare console. The result is that although the user can see the console of his machine, as provided by the VMM, he cannot "escape" and use VNC's X11 session to run other programs.

The stop VNC function takes an input a **uuid**. It calls **vncserver** to kill the desktop with the given **uuid**.

6.2.5 VMWare disks

The provider system uses VMWare (Specifically VMWare GSX Server) as the underlying VMM. Hard disks in VMWare are configured in the same way that true physical disks are. Hence, file sizes are not computed as straightforwardly as one would assume. In order to mimic the hardware as closely as possible VMWare uses cylinder, sector, and head counts to calculate disk sizes and handle disk accesses. What this means is that creating a VMWare disk requires a matching algorithm that calculates a disk size as close as possible to the size desired.

The size of the disk is

$$Size\ in\ bytes = (bytes/sector)(sectors/head)(\#\ of\ heads)(\#\ of\ cylinders)$$

$(bytes/sector)$ is generally a constant of 512 for physical disks, and VMWare assumes this as well. To get the disk size in MBytes simply divide the result by 1048576 (byes/MB).

Solving for 3 unknowns can be tricky, and there are several bounds issues and specific allowed values to take into account with the sectors and heads as well. However there are several assumptions and observations that can be made to simplify this. First, when the disk size is $>$ 2 GB, VMWare sets the head count to 255 and the sector ratio to 63. In this way the cylinder count solely determines the disk size. If we assume that all disks are going to be larger than 2 GB, which today is a safe assumption, the equation is simplified to solving for one variable, the cylinder count.

Given a request for an $SIZE$ MB disk, provided $SIZE > 2048$ (2 GB), the values needed for a VMWare disk file can be calculated as:

$$head\_count = 255$$

$$sector\_ratio = 63$$

$$sector\_count = (SIZE \times 1048576)/512$$

$$cylinder\_count = sector\_count/(head\_count \times sector\_ratio)$$

$$cylinder\_count = \lfloor cylinder\_count \rfloor$$

$$true\_disk\_size = cylinder\_count \times head\_count \times sector\_ratio \times 512 \times 1048576$$

VMWare supports several different schemes for disks. Plain disks are the simplest type, but several of their features are beneficial to Virtuoso. Plain disks are preallocated, meaning that the disk files do not automatically grow or shrink as needed. This allows for tight control on disk space use. Preallocation does mean that transfers and copies of disk files require the movement of unused data blocks. However during initialization each disk file is written with zeros so compression algorithms are very effective.

Plain disks are represented by VMWare as a text file that contains the configuration details of the disk and a set of data files whose number depends on the virtual disk size. Because the maximum file size

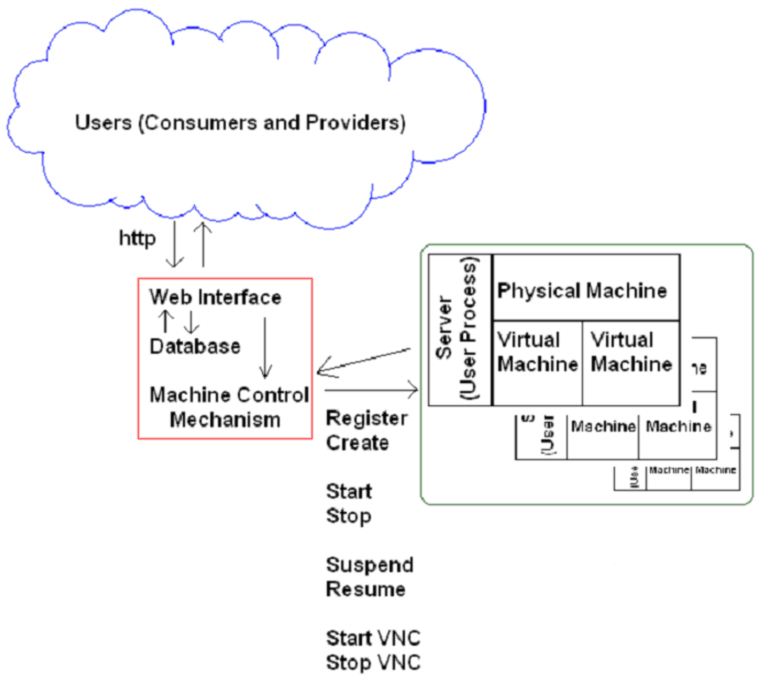


Figure 36: Overview of Virtuoso system and control interface between front-end and provider systems.

commonly supported is only 2 GB, the virtual disk is broken into 2 GB chunks. The configuration file keeps track of these chunks. When a chunk is created, it is written with zeros. The naming scheme usually follows a simple convention. $disk\_name$ is the configuration file, while $disk\_namei.dat$ is the ith chunk.

The configuration file contains information that VMWare needs to access data in the file. The cylinder count, head count, and sector ratio are all given in the file. Additionally, there is the full byte capacity of the drive, which currently does not appear to be used by VMWare. The configuration file data is tailed by the chunk listings that provide spatial information for each file. Each chunk has exactly one entry in the config file that lists it's filename, starting offset, and length.

An example config file for the disk “test” follows:

```
DRIVETYPE     scsi
#vm|VERSION           2
#vm|TOOLSVERSION      0
CYLINDERS    382
HEADS        255
SECTORS       63
#vm|CAPACITY     6136830
ACCESS "test1.dat" 0 4176900
ACCESS "test2.dat" 4176900 1959930
```

6.3 Virtual machine control interface between front-end and provider

Any user action that results in the state change of a stored machine needs to be communicated to the provider system so that it may be carried out. Figure 36 shows where the interface fits into the Virtuoso system.

Most virtual machine control commands from the front-end to the provider system are issued by the **virt_interface.pm** module located in the **server_interface** directory.

6.3.1 Storing/registering a machine

There are two paths that Virtuoso can take in storing a machine—which one is chosen depends on the type of machine which was registered:

- If the registered configuration matches the profile of a preconfigured machine, then the system performs the actions of moving that preconfigured machine from local storage to the destination machine. This involves the module **preconfigured_control.pm** calling **addpreconfigured.pl** in the **server_interface** directory. This script is a command line tool that manages the movement of a preconfigured machine to the provider on which it should be stored. This movement is separate from the migration mechanism but sets the status of the machine to **migrating** so that the user is prevented from making any changes while the machine is being stored.
- If the registered machine is not preconfigured, then the front-end passes a **create** message to the provider computer. The message contains the configuration information for the machine in an array of key : value pairs. If the create command returns a success, then the front-end passes a **register** machine message to the provider system. This message contains the uuid for the machine that is being registered with VMWare.

6.3.2 Starting/resuming a machine

If a machine is stopped, it may be started. If it is suspended, it may be resumed. The messages for starting and resuming are similar and included the **uuid** of the machine. The start command occurs before the start vnc command.

6.3.3 Starting/stopping a VNC session

As soon as a start command is successful, the system will want to start a VNC session on the provider. The first step is to set a password, this involves sending a passwd message which includes the uuid and password for the VNC session. Next, the system sends a message to connect the VNC session, this message includes a **uuid**. The provider system passes back a response that includes the port on which the VNC session has been started.

6.3.4 Stopping/suspending a machine

If a machine is running, it maybe either be suspended or stopped. Both of these commands take only a **uuid**. The fact that a machine is running when these commands are issued mean that they are prone to failure—the failure reverts the state to running. This is because VMWare may refuse to stop or suspend a machine if its operating system is still loading.

6.4 Buyer system

Currently, the buyer system consists of a java-enabled web browser.

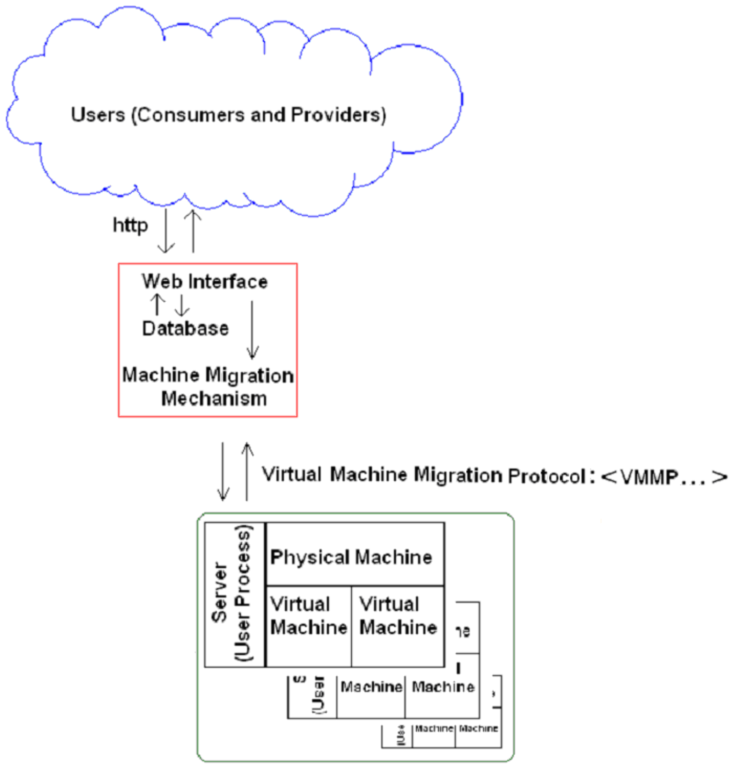

Users (Consumers and Providers)
http
Web Interface
Database
Machine Migration Mechanism
Virtual Machine Migration Protocol: <VMMP...>
Server (User Process)
Physical Machine
Virtual Machine
Virtual Machine

Figure 37: Overview of Virtuoso system and migration interface between front-end and provider systems.

7 Migration system

The motivation for including migration in Virtuoso was in part an economic one; without migration, both the provider of resources and the consumer of resources were stuck with the agreement that they had made, arbitrated by Virtuoso, at the time of machine creation. If the consumer became unhappy with the performance of their machine or the reliability of the provider's computer, they were not able to pick up and move elsewhere. Similarly, if they had found a better deal given by another provider, they were not able to take advantage of the deal. From the provider's side, this was also a disadvantage—they were not able to entice existing users to move to their system. Thus, there was not as strong of a motivation to provide competitive prices since the only customers they could get would be those that are creating a new machine and would be stuck with the chosen price. Also, a buyer would spend an inordinate amount of time looking for the best deal instead of picking a decent deal and getting on with using the machine. Migration helps to ease this friction.

Automating performance optimization and economic optimization are research topics for our group.

Figure 37 shows where the migration system fits into the Virtuoso system—it is an interface between the front-end and provider systems. The figure illustrates that all communications between the two systems happen through a Virtual Machine Migration Protocol (VMMP).

7.1 Requirements

The design of the migration system began with an enumeration of the requirements:

1. independence from any particular virtual machine platform or host platform
2. generality and interoperability
3. efficiency of migration
4. management of multiple migrations by multiple users
5. machine encapsulation
6. concurrency of versions of the machine
7. atomicity of migration transactions.

7.1.1 Independence

The goal of the Virtuoso system is to be a general purpose tool that can be inserted in-between consumers and providers; a tool that is as divorced as possible from any particular instance of a technology. This independence has two aspects; the system should not depend on any particular host operating system, nor should it depend on any particular virtual machine monitor. That is, the system may currently be built upon Red Hat Linux and use VMWare as the virtual machine monitor, but it should be possible to use any other combination of OS and VMM with minimal changes to the system.

7.1.2 Generality

Not only should the system be divorced from any particular instance of an OS or VMM, but its pieces should be as thinly linked as possible and their communication protocols well defined. Thus, the system would be general enough that new developers could come along and develop their own versions of the system that are interoperable with earlier versions—versions which could be added without complication to other instances of the system.

7.1.3 Efficiency

The fact that a virtual machine can be on the order of many gigabytes in size should not be a limiting factor in whether a user wants to migrate the machine or not. Thus, the system should take advantage of efficient migration mechanisms for the transfer of large chunks of data.

This efficiency should not come at the expense of any of the other migration requirements. Previous efforts at migration have used mechanisms which were specific to a virtual machine for example, the use of a VMWare C-shim library to capture disk writes for the creation of redo logs **(Rosenblum [10]).** This is a breach of independence between the migration system and the virtual machine monitor.

7.1.4 Machine encapsulation

Previous work has been done in building migration systems, often taking advantage of the fact that not all of the data needs to be sent to migrate a machine. Specifically, data can be fetched only when it is needed—for example on-demand paging **(Rosenblum [10, 7]).** However, this increases how much the system relies on the operation of all of its nodes and decreases overall reliability. If possible, a migration system should attempt to keep all of the machine in as concentrated an area as possible—that is, the machine should be encapsulated.

7.1.5 Consistency of versions

In the Virtuoso scheme, the machine goes through a series of states. Each state change marks a progression of the machine from an earlier version to a new version. A state change occurs when a machine is registered, created, stopped, suspended, or arrives at a migration destination. The state of the machine is stored locally with the machine and is also stored by the Virtuoso system. Thus, there is an important piece of state information that is mirrored and must be updated concurrently. The system must be designed to ensure that the version numbers are never out of synchronization.

7.1.6 Atomicity of migration

Migration is an especially critical period of state change. That is, there is a chance that something might go wrong and that the migration is not a success. The situation where a machine is left in a state of transition should not be a terminal one, that is we must be able to know that a migration was not successful and we must be able to revert to the situation the machine was in right before migration.

7.2 Migration implementation

The implementation consists of a mechanism-independent migration protocol, a migration agent and interface that implement the protocol, and ssh and rsync transfer methods.

7.2.1 Migration protocol

The requirement of generality and interoperability led to the definition of a Virtual Machine Migration Protocol (VMMP). The Virtuoso system supports version 1.0 of this protocol.

Through the use of VMMP, the migration agent on the central server only needs to know how to communicate with the migration server on the destination machine. The central server issues queries and requests and gets results and responses from the destination machines.

There are five supported VMMP messages:

1. <VMMP 1.0 QUERY MACHINE_ID METHODS>
2. <VMMP 1.0 RESULTS MACHINE_ID N_METHODS METHOD_1 METHOD_2 ... METHOD_N>
3. <VMMP 1.0 REQ MACHINE_ID VERSION ORIG_IP ORIG_PORT CENTRAL_IP CENTRAL_PORT METHOD>
4. <VMMP 1.0 HEARTBEAT MACHINE_ID>
5. <VMMP 1.0 RESP MACHINE_ID VERSION' SUCCESS/FAIL>

The QUERY message is used to ask the providers' machines which methods of migration they support. The MACHINE_ID is the universal unique identifier of the virtual machine, as stored in the database. The 1.0 in front of VMMP is the version of the protocol that we are using.

If the QUERY is received and if the destination machine supports protocol version 1.0, then the response that the machine produces is a RESULTS message. The tag N_METHODS is an integer which marks how many methods it has returned. The instance of the migration system that has been built supports the methods of RSYNC and SCP—the merits of these methods and how they are used are discussed later.

The REQ message is a request to the server on the destination machine to migrate the machine MACHINE_ID using the specified method (METHOD). The ORIG_IP and ORIG_PORT are the values of the address of the host on which the machine is currently stored and of the port on which the migration server on that machine is listening.

The HEARBEAT message is a message generated by the destination machine intermittently to let the central server know that the destination machine is still running and is still migrating the machine that has been requested. It is sent to CENTRAL_IP/CENTRAL_PORT and is the reason that those two values are sent with the REQ message. If the heartbeat is not received within a pre-specified window time, the central server decides that the migration failed and reverts the system to its previous state. This is one way through which atomicity is approached—the system makes sure that any failure is regarded as a complete failure and an earlier state is reinstated.

The RESP message is sent by the destination server once it has received the machine and has checked that its local state is sane. The central server can then update the state of the machine to VERSION', and update other appropriate database tables: machine pairings table with the new machine location, and physical machine table to reflect the memory, storage and speed changes.

7.2.2 Migration agent

The migration agent is a bridge between the web interface and the Virtuoso provider server on the destination machine. All requests for migration and migration related queries come through the migration agent on the central server in the form of VMMP requests.

The agent is a Perl script that runs in a continuous loop, waiting for requests. If a request is for a QUERY message, it sends the message directly to the destination machine and returns the result via an open socket with the web interface. If, however, the message is for a REQ, the migration agent cannot afford to wait for a response from the server on the destination machine. Instead, it forks an independent server that is charged with handling that specific request for that machine id. The migration agent maintains a list of ports on which migration servers are already running. When a server finishes, the server cleans the port list of the port it was using. The migration ports, by policy, start at 10000.

7.2.3 Migration server on provider machine

On the destination machine, there is a server running **virtuoso.pl** that is responsible for both regular machine operating requests and for migration requests. It supports many different messages: VMMP messages, machine operating messages, and back-end specific messages.

The VMMP messages have already been discussed, as have the machine operating messages and vnc operating messages (start/stop/suspend/resume/register/passwd/ connect/disconnect/create).

The back-end messages are those messages that are used by the origin and destination machines to communicate with each other during a migration event. The messages that are supported are:

register_pre_configured This function is a necessity brought about by VMWare—a machine cannot simple be started with VMWare, even though it has been created. Upon the termination of migration—the machine must be registered.

ask_virt_path ask_base_path ask_config_path These functions are used by the copying methods to copy the correct directory to the correct destination directory.

make_vm_migratable This function is used to make sure that the permissions of the files used by the virtual machine are set correctly so a copy is permitted to the Virtuoso user. In further extensions to the system, this function will probably be more important as it is a general purpose function to make sure that the machine state is sane for migration.

ask_state This is a function which figures out the state of the virtual machine.

There are many local modules that the migration server uses. For migration, the modules are:

vmmp_back This module executes VMMP instructions on behalf of **virtuoso.pl.**

The lookup modules

- **back_end_lookup** This is a wrapper for communications from the destination server to the origin server. This module is used by [method]_server.pl to ask the origin machine where it has stored the machine with a given machine id.
- **vm_lookup** This is used to look up path information for the virtual machines stored on a physical machine—it accesses the virt_config.pm module to see where the machines are stored.
- **vm_version_lookup vm_origin_lookup** These two modules look up the version file list and origin lookup list respectively. These lists keep the information for the version stored on the machine and for the last location of a given virtual machine.

(scp — rsync)_server.pl These are wrappers for the given copying methods (scp and rsync). These are currently the two methods supported by the system. They are command line tools used by vmmp_back to execute REQ instructions.

vm_control.pm This is a wrapper for VMWare's Perl API—in the migration context, it is used to register machines once the migration process is completed.

heartbeat_server.pl This script reports intermittently to the central server by sending VMMP HEARTBEAT messages.

7.2.4 Local state

In addition to the database, there are local state files that are maintained by the host machines. The ones which are important to the migration system are the **vm_list.virt**, **vm_origin_list.virt**, and **vm_version_list.virt** files. These are newline delimited machineid : value pair files. They are parsed by their corresponding lookup functions.

7.2.5 Supported methods

Currently the migration system supports two copying methods. These are rsync (an updated version of rcp, which implements a remote update protocol based on block hash comparisons) and scp, which does an straight copy over an encrypted channel. These two methods were chosen because they are well behaved and contrast each other enough to give interesting performance data.

The advantages of rsync are that it does compression and updates to files—it doesn't send a block if it is already at the destination. Scp does a simple transfer and should provide a baseline of performance. Their relative performances are discussed in Section 7.4.

We have been worked on a migration method based on a versioning file system, but it is not ready yet.

7.3 Usage scenario

Here we describe how the migration process works from mouse click in the web interface to completion, all at the granularity of a migration log:

7.3.1 Front-end initiates migration and waits

The user clicks migrate on the migrate.pl page.

- **startmigration.pl** has to check the sanity of the request:
  1. Checks the database to make sure the user has permission to start the machine with the given machine id.
  2. Uses **vmmp.pm** to QUERY the destination for which methods it supports.
  3. **virtuoso.pl** on the provider destination machine receives the query and consults **vmmp_back.pm** to ask the methods that it supports.
  4. startmigration.pl uses vmmp.pm to parse the results and determines that the destination supports the migration method
- The request is sane, it is ok to issue a migration request:
  1. **startmigration.pl** calls **start_migration** through the **vmmp.pm** package—sending the version, machine id, destination ip/id, origin ip/id, and method.
  2. **vmmp.pm** sends a message to the migration agent on the central server, giving it the newly constructed VMMP request and the destination information. The migration agent, passes this information off to a migration server that it starts in the background on the same machine—the migration agent keeps track of which ports the servers that it has started are using.
  3. **migration_server.pl** sends the VMMP message to **virtuoso.pl** on the destination machine, it now waits for a response in the form of either a HEARTBEAT or a RESP. It will continue to spin until it either times out because no heartbeat arrives or the response is a SUCCESS/FAILURE.

7.3.2 Providers coordinate and perform migration

1. **virtuoso.pl** on the destination machine receives the request and uses **vmmp_back.pm** to execute the request.

2. **vmmp_back** on the destination machine unregisters the machine locally by deleting its entry from the **vm_list.virt** file, if the machine was already cached there.

3. Depending on the METHOD type, **vmmp_back.pm** starts the appropriate server (in this case, either **scp_server.pl** or **rsync_server.pl**).

4. **virtuoso.pl** upon receipt of a **migrating** message from **vmmp_back** starts a heartbeat server on the destination machine.

5. **heartbeat_server.pl** intermittently checks the local **vm_list.virt** file to see if the machine with the given machine id exists, if it exists, it knows that the machine migration has been successful.

6. In the meantime, the appropriate method of migration is being executed (either scp or rsync in this case) by the appropriate server.

7. **(scp — rsync)_server.pl** sends messages to the origin machine to inquire as to the information it has about the machine with the given machine id. Namely, it wants to know the machine's absolute directory, the absolute path to the configuration file, and the state of the machine. Then, it issues the command **make_vm_migratable** which tells the origin that it is about to receive a command to migrate, so the origin should make sure that the permissions are set properly to allow for migration.

8. **(scp — rsync)_server.pl** then does a system call to start the specified method to copy the machine into their local Virtuoso directories.

9. When the copy is completed, the local files are updated with the new version of the machine, the path to its configuration file, and the machine from which it was copied. Also, the VMWare Perl API is invoked to register the machine with VMWare.

10. Now that the machine has been locally registered, the heartbeat server notices that the machine exists in **vm_list.virt** and returns the SUCCESS response to the migration server on the central server.

7.3.3 Front-end is updated and updates database

- **migration_server.pl** receives a response from the destination:

  1. If the response is a SUCCESS, then it will update the database in the following ways: the version will be set to the returned version, the memory, storage and speed of the destination and origin machines will be decreased and increased respectively, the machine pairings will be updated with the new pair of machine id and physical machine id, and the state of the machine will be returned from ”migrating” to its previous state. Also the user account will be debited and the provider account credited.
  2. If the response is a FAILURE then it will revert the system to the previous state, which the script keeps in local variables.

7.4 Migration performance

In the previous section, an in-depth account of the migration sequence was given. Although it is a fairly long sequence of events, the major bottleneck occurs between steps 8 and 9, between when the copy command is issued and when it terminates. In measuring the performance of the system, this was the only metric that was used; how long the actual copy took.

The first test that was performed on the system was a test to see how long it would take to migrate a suspended machine (a 1.1 Gigabyte win2000Pro machine over a 100Mbit network) when the machine was run without disk use, when it was run with disk use, and when it was run with heavy disk use. The methods of rsync and scp were compared in this test.

Figure 38 shows the results for rsync and Figure 39 shows the results for scp. Note that these tests were done at different times and it is not the height of the graphs that is important but rather their relative heights to each other. That is, scp is not necessarily 6 times slower than rsync in copying an entire disk without caching, but it is much slower than rsync after an older copy already exists on the destination machines. In test 1 of scp and rsync, VMWare’s undoable disk was used. A machine was run on a single machine and then suspended. It was then sent to 6 machines in a round-robin fashion; these machines did not have a previous version of that virtual machine available.

The machine was then sent around again to measure the latency of rsync in checking the file lists for differences and sending the differences—in this case, there were none. The machine was then run and suspended—the time spent is mainly the time it took to send VMWare’s state file (.vmss), which is mainly memory contents. The “ballooning” technique described in [10] would potentially help to reduce these costs.

The next two tests used the disk[28] slightly—the second test being an automated one.

[28]VMWare’s undoable disk option actually writes to a redolog file

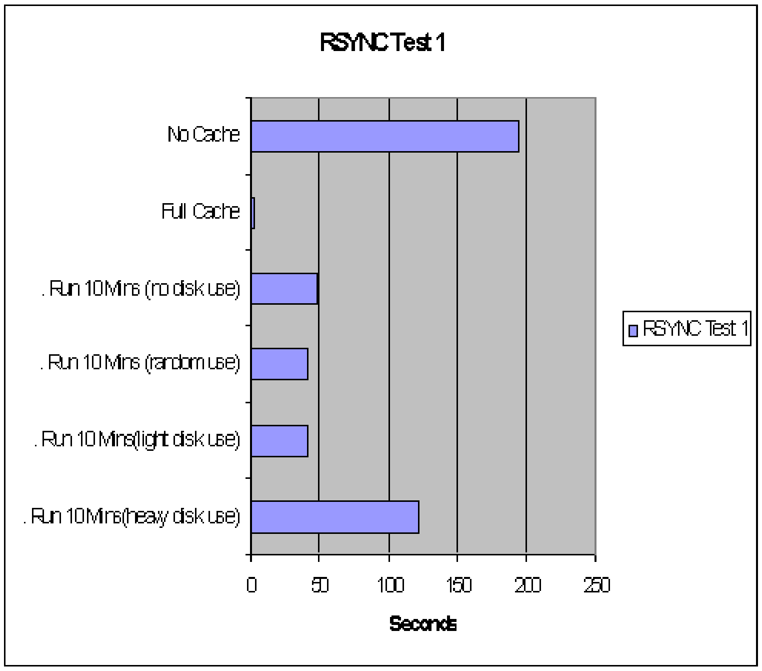


Figure 38: First rsync test.

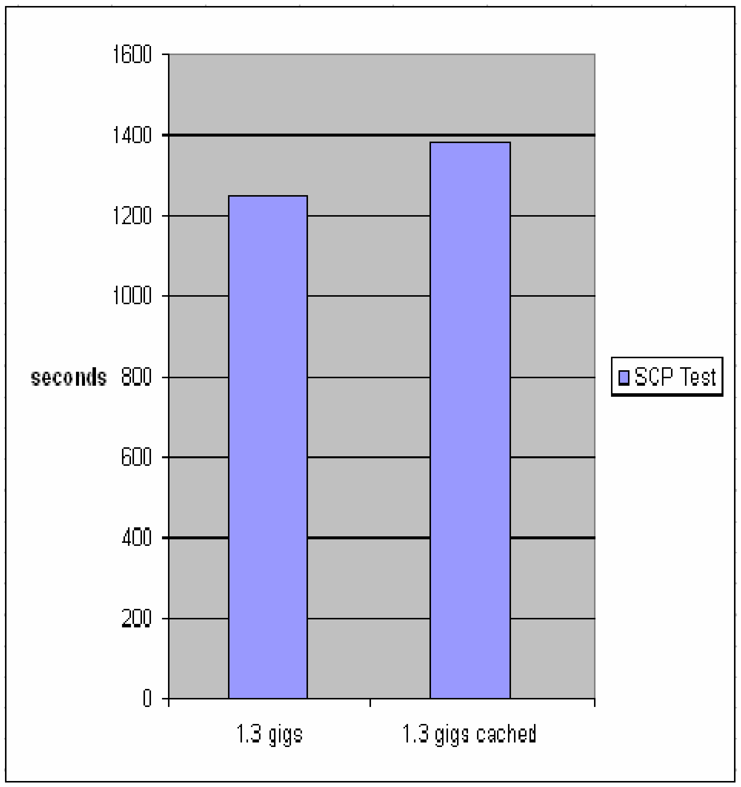


Figure 39: SCP test.

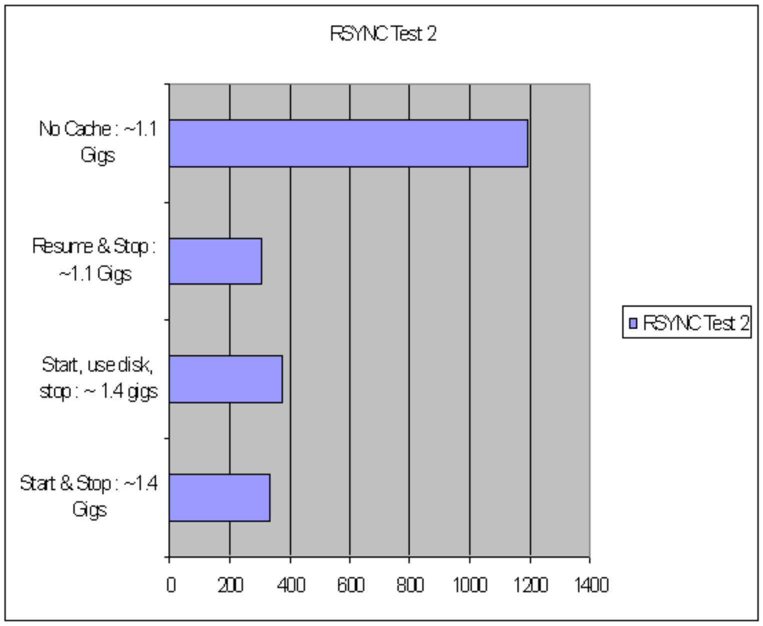


Figure 40: Second rsync test.

The final test was one where the disk was updated heavily—about 250 megabytes were written to the disk in the space of 10 minutes. In each case, the machine was suspended and sent around the six nodes.

The test for scp was not as thorough as the one for rsync as it became obvious that no performance difference would be seen upon further testing.

The results show that rsync is fairly efficient for transfering VMs. The fact that the average time to migrate a fully cached machine was less than 5 seconds means that a user is now free to migrate their machine frequently in a search for a better provider.

Seeing that scp was consistent in its migration times, a second test was not done on it. However, rsync was explored further—a second test was run on it to see what happened when the redolog files generated by VMWare were committed to disk. The results of this test are seen in Figure 40.

As before, in the first case, a suspended machine was sent to a machine on which it was not cached. In the second case, the machine was resumed and then VMWare was forced to commit the VMWare redolog files to disk—which amounted to the writing of about fifty megabytes of data to the disk, and the deletion of about fifty megabytes of redolog files. In the third test, the disk was started and about 250 megabytes of data were written to it. In the final test, the machine was simply started and stopped—as before the redo logs were committed to disk. It appears from this test that there is a fixed overhead of writing to a disk—this is probably a side effect of the operating system running. Compared to this, the actual writing to disk seems to be a small overhead. This result makes sense, especially if most of the writes happened to a single section of disk—since this is the situation that rsync was designed for.

8 Conclusion and future work

We have described in detail the interface, design, and implementation of the Virtuoso system. The examples provided have thoroughly shown how the system can be fully used by both providers and buyers. The system is useful prototype that furthers the goal of helping to bring providers of resources together with buyers of resources, to their mutual benefit. We provided an in-depth description of the implementation which should be sufficient as a technical reference for developers in the Virtuoso system.

We are currently integrating Virtuoso, as described here, with the VNET virtual network system. We also plan to integrate remote device support to make configuring a VM straightforward. Our research agenda is moving towards adaptation using Virtuoso and VNET. In particular, a future version of this system will automatically determine VM to provider mappings, as well as virtual network topology and routing, and resource reservations on the underlying hardware, to optimize the performance of groups of VMs running parallel and distributed applications.

Please visit virtuoso.cs.northwestern.edu to learn more.

References

[1] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *ACM Symposium on Operating Systems Principles (SOSP)* (2003), pp. 164–177.

[2] CORNELL, B., DINDA, P., AND BUSTAMANTE, F. Wayback: A user-level versioning file system for linux. In *Proceedings of USENIX 2004 (Freenix Track)* (July 2004). To Appear.

[3] FIGUEIREDO, R., DINDA, P. A., AND FORTES, J. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)* (May 2003).

[4] GOLDBERG, R. P. Survey of virtual machine research. *IEEE Computer 7,* 6 (1974), 34–45.

[5] GUPTA, A., AND DINDA, P. A. Inferring the topology and traffic load of parallel programs running in a virtual machine environment. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing (JSPPS 2004* (June 2004). To Appear.

[6] GUPTA, A., LIN, B., AND DINDA, P. A. Measuring and understanding user comfort with resource borrowing. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC 2004)* (June 2004). To Appear.

[7] KOZUCH, M., AND SATYANARAYANAN, M. Internet suspend/resume. In *Proceedings of the 4th Workshop on Mobile Computing Systems and Applications* (June 2002).

[8] LINUX FREEVSD PROJECT. http://www.freevsd.org.

[9] LINUX VSERVER PROJECT. http://www.linux-vserver.org.

[10] SAPUNTZAKIS, C., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M., AND ROSENBLUM, M. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)* (2002).

[11] SMITH, J. E. An overview of virtual machine architectures. Department of Electrical and Computer Engineering, University of Wisconsin, 2001.

[12] SUNDARARAJ, A., AND DINDA, P. Exploring inference-based monitoring of virtual machine resources. Tech. rep., Department of Computer Science, Northwestern University, 2004.

[13] SUNDARARAJ, A., AND DINDA, P. Towards virtual networks for virtual machine grid computing. In *Proceedings of the 3rd USENIX Virtual Machine Research And Technology Symposium (VM 2004)* (May 2004). To Appear. Earlier version available as Technical Report NWU-CS-03-27, Department of Computer Science, Northwestern University.

[14] WHITAKER, A., SHAW, M., AND GRIBBLE, S. Scale and performance in the denali isolation kernel. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI 2002)* (December 2002).