Users' Manual

Xen v3.0

DISCLAIMER: This documentation is always under active development and as such there may be mistakes and omissions — watch out for these and please report any you find to the developers' mailing list, xen-devel@lists.xensource.com. The latest version is always available on-line. Contributions of material, suggestions and corrections are welcome.

Contents

Chapter 1

Introduction

Xen is an open-source *para-virtualizing* virtual machine monitor (VMM), or "hypervisor", for the x86 processor architecture. Xen can securely execute multiple virtual machines on a single physical system with close-to-native performance. Xen facilitates enterprise-grade functionality, including:

- Virtual machines with performance close to native hardware.
- Live migration of running virtual machines between physical hosts.
- Up to 32 virtual CPUs per guest virtual machine, with VCPU hotplug.
- x86/32, x86/32 with PAE, and x86/64 platform support.
- Intel Virtualization Technology (VT-x) for unmodified guest operating systems (including Microsoft Windows).
- Excellent hardware support (supports almost all Linux device drivers).

1.1 Usage Scenarios

Usage scenarios for Xen include:

Server Consolidation. Move multiple servers onto a single physical host with performance and fault isolation provided at the virtual machine boundaries.

Hardware Independence. Allow legacy applications and operating systems to exploit new hardware.

Multiple OS configurations. Run multiple operating systems simultaneously, for development or testing purposes.

Kernel Development. Test and debug kernel modifications in a sand-boxed virtual machine — no need for a separate test machine.

Cluster Computing. Management at VM granularity provides more flexibility than

separately managing each physical host, but better control and isolation than single-system image solutions, particularly by using live migration for load balancing.

Hardware support for custom OSes. Allow development of new OSes while benefiting from the wide-ranging hardware support of existing OSes such as Linux.

1.2 Operating System Support

Para-virtualization permits very high performance virtualization, even on architectures like x86 that are traditionally very hard to virtualize.

This approach requires operating systems to be *ported* to run on Xen. Porting an OS to run on Xen is similar to supporting a new hardware platform, however the process is simplified because the para-virtual machine architecture is very similar to the underlying native hardware. Even though operating system kernels must explicitly support Xen, a key feature is that user space applications and libraries *do not* require modification.

With hardware CPU virtualization as provided by Intel VT and AMD SVM technology, the ability to run an unmodified guest OS kernel is available. No porting of the OS is required, although some additional driver support is necessary within Xen itself. Unlike traditional full virtualization hypervisors, which suffer a tremendous performance overhead, the combination of Xen and VT or Xen and Pacifica technology complement one another to offer superb performance for para-virtualized guest operating systems and full support for unmodified guests running natively on the processor. Full support for VT and Pacifica chipsets will appear in early 2006.

Paravirtualized Xen support is available for increasingly many operating systems: currently, mature Linux support is available and included in the standard distribution. Other OS ports—including NetBSD, FreeBSD and Solaris x86 v10—are nearing completion.

1.3 Hardware Support

Xen currently runs on the x86 architecture, requiring a "P6" or newer processor (e.g. Pentium Pro, Celeron, Pentium II, Pentium III, Pentium IV, Xeon, AMD Athlon, AMD Duron). Multiprocessor machines are supported, and there is support for HyperThreading (SMT). In addition, ports to IA64 and Power architectures are in progress.

The default 32-bit Xen supports up to 4GB of memory. However Xen 3.0 adds support for Intel's Physical Addressing Extensions (PAE), which enable x86/32 machines to address up to 64 GB of physical memory. Xen 3.0 also supports x86/64 platforms such

as Intel EM64T and AMD Opteron which can currently address up to 1TB of physical memory.

Xen offloads most of the hardware support issues to the guest OS running in the *Domain 0* management virtual machine. Xen itself contains only the code required to detect and start secondary processors, set up interrupt routing, and perform PCI bus enumeration. Device drivers run within a privileged guest OS rather than within Xen itself. This approach provides compatibility with the majority of device hardware supported by Linux. The default XenLinux build contains support for most server-class network and disk hardware, but you can add support for other hardware by configuring your XenLinux kernel in the normal way.

1.4 Structure of a Xen-Based System

A Xen system has multiple layers, the lowest and most privileged of which is Xen itself.

Xen may host multiple *guest* operating systems, each of which is executed within a secure virtual machine. In Xen terminology, a *domain*. Domains are scheduled by Xen to make effective use of the available physical CPUs. Each guest OS manages its own applications. This management includes the responsibility of scheduling each application within the time allotted to the VM by Xen.

The first domain, *domain 0*, is created automatically when the system boots and has special management privileges. Domain 0 builds other domains and manages their virtual devices. It also performs administrative tasks such as suspending, resuming and migrating other virtual machines.

Within domain 0, a process called *xend* runs to manage the system. Xend is responsible for managing virtual machines and providing access to their consoles. Commands are issued to xend over an HTTP interface, via a command-line tool.

1.5 History

Xen was originally developed by the Systems Research Group at the University of Cambridge Computer Laboratory as part of the XenoServers project, funded by the UK-EPSRC.

XenoServers aim to provide a "public infrastructure for global distributed computing". Xen plays a key part in that, allowing one to efficiently partition a single machine to enable multiple independent clients to run their operating systems and applications in an environment. This environment provides protection, resource isolation and accounting. The project web page contains further information along with pointers to papers and technical reports: `http://www.cl.cam.ac.uk/xeno`

Xen has grown into a fully-fledged project in its own right, enabling us to investigate interesting research issues regarding the best techniques for virtualizing resources such as the CPU, memory, disk and network. Project contributors now include XenSource, Intel, IBM, HP, AMD, Novell, RedHat.

Xen was first described in a paper presented at SOSP in 2003[1], and the first public release (1.0) was made that October. Since then, Xen has significantly matured and is now used in production scenarios on many sites.

1.6 What's New

Xen 3.0.0 offers:

- Support for up to 32-way SMP guest operating systems
- Intel (Physical Addressing Extensions) PAE to support 32-bit servers with more than 4GB physical memory
- x86/64 support (Intel EM64T, AMD Opteron)
- Intel VT-x support to enable the running of unmodified guest operating systems (Windows XP/2003, Legacy Linux)
- Enhanced control tools
- Improved ACPI support
- AGP/DRM graphics

Xen 3.0 features greatly enhanced hardware support, configuration flexibility, usability and a larger complement of supported operating systems. This latest release takes Xen a step closer to being the definitive open source solution for virtualization.

[1] `http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf`

Part I

Installation

Chapter 2

Basic Installation

The Xen distribution includes three main components: Xen itself, ports of Linux and NetBSD to run on Xen, and the userspace tools required to manage a Xen-based system. This chapter describes how to install the Xen 3.0 distribution from source. Alternatively, there may be pre-built packages available as part of your operating system distribution.

2.1 Prerequisites

The following is a full list of prerequisites. Items marked '†' are required by the xend control tools, and hence required if you want to run more than one virtual machine; items marked '∗' are only required if you wish to build from source.

- A working Linux distribution using the GRUB bootloader and running on a P6-class or newer CPU.

† The `iproute2` package.

† The Linux bridge-utils[1] (e.g., `/sbin/brctl`)

† The Linux hotplug system[2] (e.g., `/sbin/hotplug` and related scripts). On newer distributions, this is included alongside the Linux udev system[3].

∗ Build tools (gcc v3.2.x or v3.3.x, binutils, GNU make).

∗ Development installation of zlib (e.g., zlib-dev).

∗ Development installation of Python v2.2 or later (e.g., python-dev).

∗ LaTeX and transfig are required to build the documentation.

[1] Available from `http://bridge.sourceforge.net`
[2] Available from `http://linux-hotplug.sourceforge.net/`
[3] See `http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html/`

Once you have satisfied these prerequisites, you can now install either a binary or source distribution of Xen.

2.2 Installing from Binary Tarball

Pre-built tarballs are available for download from the XenSource downloads page:

`http://www.xensource.com/downloads/`

Once you've downloaded the tarball, simply unpack and install:

```
# tar zxvf xen-3.0-install.tgz
# cd xen-3.0-install
# sh ./install.sh
```

Once you've installed the binaries you need to configure your system as described in Section 2.5.

2.3 Installing from RPMs

Pre-built RPMs are available for download from the XenSource downloads page:

`http://www.xensource.com/downloads/`

Once you've downloaded the RPMs, you typically install them via the RPM commands:

```
# rpm -iv rpmname
```

See the instructions and the Release Notes for each RPM set referenced at:

`http://www.xensource.com/downloads/`.

2.4 Installing from Source

This section describes how to obtain, build and install Xen from source.

2.4.1 Obtaining the Source

The Xen source tree is available as either a compressed source tarball or as a clone of our master Mercurial repository.

Obtaining the Source Tarball

Stable versions and daily snapshots of the Xen source tree are available from the Xen download page:

`http://www.xensource.com/downloads/`

Obtaining the source via Mercurial

The source tree may also be obtained via the public Mercurial repository at:

```
http://xenbits.xensource.com
```

See the instructions and the Getting Started Guide referenced at:

```
http://www.xensource.com/downloads/
```

2.4.2 Building from Source

The top-level Xen Makefile includes a target "world" that will do the following:

- Build Xen.
- Build the control tools, including xend.
- Download (if necessary) and unpack the Linux 2.6 source code, and patch it for use with Xen.
- Build a Linux kernel to use in domain 0 and a smaller unprivileged kernel, which can be used for unprivileged virtual machines.

After the build has completed you should have a top-level directory called `dist/` in which all resulting targets will be placed. Of particular interest are the two XenLinux kernel images, one with a "-xen0" extension which contains hardware device drivers and drivers for Xen's virtual devices, and one with a "-xenU" extension that just contains the virtual ones. These are found in `dist/install/boot/` along with the image for Xen itself and the configuration files used during the build.

To customize the set of kernels built you need to edit the top-level Makefile. Look for the line:

```
KERNELS ?= linux-2.6-xen0 linux-2.6-xenU
```

You can edit this line to include any set of operating system kernels which have configurations in the top-level `buildconfigs/` directory.

2.4.3 Custom Kernels

If you wish to build a customized XenLinux kernel (e.g. to support additional devices or enable distribution-required features), you can use the standard Linux configuration mechanisms, specifying that the architecture being built for is `xen`, e.g:

```
# cd linux-2.6.12-xen0
# make ARCH=xen xconfig
# cd ..
# make
```

You can also copy an existing Linux configuration (`.config`) into e.g. `linux-2.6.12-xen0` and execute:

```
# make ARCH=xen oldconfig
```

You may be prompted with some Xen-specific options. We advise accepting the defaults for these options.

Note that the only difference between the two types of Linux kernels that are built is the configuration file used for each. The "U" suffixed (unprivileged) versions don't contain any of the physical hardware device drivers, leading to a 30% reduction in size; hence you may prefer these for your non-privileged domains. The "0" suffixed privileged versions can be used to boot the system, as well as in driver domains and unprivileged domains.

2.4.4 Installing Generated Binaries

The files produced by the build process are stored under the `dist/install/` directory. To install them in their default locations, do:

```
# make install
```

Alternatively, users with special installation requirements may wish to install them manually by copying the files to their appropriate destinations.

The `dist/install/boot` directory will also contain the config files used for building the XenLinux kernels, and also versions of Xen and XenLinux kernels that contain debug symbols such as (`xen-syms-3.0.0` and `vmlinux-syms-2.6.12.6-xen0`) which are essential for interpreting crash dumps. Retain these files as the developers may wish to see them if you post on the mailing list.

2.5 Configuration

Once you have built and installed the Xen distribution, it is simple to prepare the machine for booting and running Xen.

2.5.1 GRUB Configuration

An entry should be added to `grub.conf` (often found under `/boot/` or `/boot/grub/`) to allow Xen / XenLinux to boot. This file is sometimes called `menu.lst`, depending on your distribution. The entry should look something like the following:

```
title Xen 3.0 / XenLinux 2.6
  kernel /boot/xen-3.0.gz dom0_mem=262144
  module /boot/vmlinuz-2.6-xen0 root=/dev/sda4 ro console=tty0
```

The kernel line tells GRUB where to find Xen itself and what boot parameters should be passed to it (in this case, setting the domain 0 memory allocation in kilobytes and

the settings for the serial port). For more details on the various Xen boot parameters see Section 10.3.

The module line of the configuration describes the location of the XenLinux kernel that Xen should start and the parameters that should be passed to it. These are standard Linux parameters, identifying the root device and specifying it be initially mounted read only and instructing that console output be sent to the screen. Some distributions such as SuSE do not require the `ro` parameter.

To use an initrd, add another `module` line to the configuration, like:

```
module /boot/my_initrd.gz
```

When installing a new kernel, it is recommended that you do not delete existing menu options from `menu.lst`, as you may wish to boot your old Linux kernel in future, particularly if you have problems.

2.5.2 Serial Console (optional)

Serial console access allows you to manage, monitor, and interact with your system over a serial console. This can allow access from another nearby system via a null-modem ("LapLink") cable or remotely via a serial concentrator.

You system's BIOS, bootloader (GRUB), Xen, Linux, and login access must each be individually configured for serial console access. It is *not* strictly necessary to have each component fully functional, but it can be quite useful.

For general information on serial console configuration under Linux, refer to the "Remote Serial Console HOWTO" at The Linux Documentation Project: `http://www.tldp.org`

Serial Console BIOS configuration

Enabling system serial console output neither enables nor disables serial capabilities in GRUB, Xen, or Linux, but may make remote management of your system more convenient by displaying POST and other boot messages over serial port and allowing remote BIOS configuration.

Refer to your hardware vendor's documentation for capabilities and procedures to enable BIOS serial redirection.

Serial Console GRUB configuration

Enabling GRUB serial console output neither enables nor disables Xen or Linux serial capabilities, but may made remote management of your system more convenient by displaying GRUB prompts, menus, and actions over serial port and allowing remote GRUB management.

Adding the following two lines to your GRUB configuration file, typically either `/boot/grub/menu.lst` or `/boot/grub/grub.conf` depending on your distro, will enable GRUB serial output.

```
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
terminal --timeout=10 serial console
```

Note that when both the serial port and the local monitor and keyboard are enabled, the text "*Press any key to continue*" will appear at both. Pressing a key on one device will cause GRUB to display to that device. The other device will see no output. If no key is pressed before the timeout period expires, the system will boot to the default GRUB boot entry.

Please refer to the GRUB documentation for further information.

Serial Console Xen configuration

Enabling Xen serial console output neither enables nor disables Linux kernel output or logging in to Linux over serial port. It does however allow you to monitor and log the Xen boot process via serial console and can be very useful in debugging.

In order to configure Xen serial console output, it is necessary to add a boot option to your GRUB config; e.g. replace the previous example kernel line with:

```
kernel /boot/xen.gz dom0_mem=131072 com1=115200,8n1
```

This configures Xen to output on COM1 at 115,200 baud, 8 data bits, no parity and 1 stop bit. Modify these parameters for your environment. See Section 10.3 for an explanation of all boot parameters.

One can also configure XenLinux to share the serial console; to achieve this append "`console=ttyS0`" to your module line.

Serial Console Linux configuration

Enabling Linux serial console output at boot neither enables nor disables logging in to Linux over serial port. It does however allow you to monitor and log the Linux boot process via serial console and can be very useful in debugging.

To enable Linux output at boot time, add the parameter `console=ttyS0` (or ttyS1, ttyS2, etc.) to your kernel GRUB line. Under Xen, this might be:

```
module /vmlinuz-2.6-xen0 ro root=/dev/VolGroup00/LogVol00 \
console=ttyS0, 115200
```

to enable output over ttyS0 at 115200 baud.

Serial Console Login configuration

Logging in to Linux via serial console, under Xen or otherwise, requires specifying a login prompt be started on the serial port. To permit root logins over serial console, the serial port must be added to `/etc/securetty`.

To automatically start a login prompt over the serial port, add the line:

```
c:2345:respawn:/sbin/mingetty ttyS0
```

to `/etc/inittab`. Run `init q` to force a reload of your inttab and start getty.

To enable root logins, add `ttyS0` to `/etc/securetty` if not already present.

Your distribution may use an alternate getty; options include getty, mgetty and agetty. Consult your distribution's documentation for further information.

2.5.3 TLS Libraries

Users of the XenLinux 2.6 kernel should disable Thread Local Storage (TLS) (e.g. by doing a `mv /lib/tls /lib/tls.disabled`) before attempting to boot a XenLinux kernel[4]. You can always reenable TLS by restoring the directory to its original location (i.e. `mv /lib/tls.disabled /lib/tls`).

The reason for this is that the current TLS implementation uses segmentation in a way that is not permissible under Xen. If TLS is not disabled, an emulation mode is used within Xen which reduces performance substantially. To ensure full performance you should install a 'Xen-friendly' (nosegneg) version of the library.

2.6 Booting Xen

It should now be possible to restart the system and use Xen. Reboot and choose the new Xen option when the Grub screen appears.

What follows should look much like a conventional Linux boot. The first portion of the output comes from Xen itself, supplying low level information about itself and the underlying hardware. The last portion of the output comes from XenLinux.

You may see some error messages during the XenLinux boot. These are not necessarily anything to worry about—they may result from kernel configuration differences between your XenLinux kernel and the one you usually use.

When the boot completes, you should be able to log into your system as usual. If you are unable to log in, you should still be able to reboot with your normal Linux kernel by selecting it at the GRUB prompt.

[4] If you boot without first disabling TLS, you will get a warning message during the boot process. In this case, simply perform the rename after the machine is up and then run `/sbin/ldconfig` to make it take effect.

Chapter 3

Booting a Xen System

Booting the system into Xen will bring you up into the privileged management domain, Domain0. At that point you are ready to create guest domains and "boot" them using the `xm create` command.

3.1 Booting Domain0

After installation and configuration is complete, reboot the system and and choose the new Xen option when the Grub screen appears.

What follows should look much like a conventional Linux boot. The first portion of the output comes from Xen itself, supplying low level information about itself and the underlying hardware. The last portion of the output comes from XenLinux.

When the boot completes, you should be able to log into your system as usual. If you are unable to log in, you should still be able to reboot with your normal Linux kernel by selecting it at the GRUB prompt.

The first step in creating a new domain is to prepare a root filesystem for it to boot. Typically, this might be stored in a normal partition, an LVM or other volume manager partition, a disk file or on an NFS server. A simple way to do this is simply to boot from your standard OS install CD and install the distribution into another partition on your hard drive.

To start the xend control daemon, type

```
# xend start
```

If you wish the daemon to start automatically, see the instructions in Section 4.1. Once the daemon is running, you can use the `xm` tool to monitor and maintain the domains running on your system. This chapter provides only a brief tutorial. We provide full details of the `xm` tool in the next chapter.

3.2 Booting Guest Domains

3.2.1 Creating a Domain Configuration File

Before you can start an additional domain, you must create a configuration file. We provide two example files which you can use as a starting point:

- `/etc/xen/xmexample1` is a simple template configuration file for describing a single VM.
- `/etc/xen/xmexample2` file is a template description that is intended to be reused for multiple virtual machines. Setting the value of the `vmid` variable on the `xm` command line fills in parts of this template.

There are also a number of other examples which you may find useful. Copy one of these files and edit it as appropriate. Typical values you may wish to edit include:

kernel Set this to the path of the kernel you compiled for use with Xen (e.g. `kernel = ''/boot/vmlinuz-2.6-xenU''`)

memory Set this to the size of the domain's memory in megabytes (e.g. `memory = 64`)

disk Set the first entry in this list to calculate the offset of the domain's root partition, based on the domain ID. Set the second to the location of `/usr` if you are sharing it between domains (e.g. `disk = ['phy:your_hard_drive%d,sda1,w' % (base_partition_number + vmid), 'phy:your_usr_partition,sda6,r' ]`

dhcp Uncomment the dhcp variable, so that the domain will receive its IP address from a DHCP server (e.g. `dhcp=''dhcp''`)

You may also want to edit the **vif** variable in order to choose the MAC address of the virtual ethernet interface yourself. For example:

```
vif = ['mac=00:16:3E:F6:BB:B3']
```

If you do not set this variable, xend will automatically generate a random MAC address from the range 00:16:3E:xx:xx:xx, assigned by IEEE to XenSource as an OUI (organizationally unique identifier). XenSource Inc. gives permission for anyone to use addresses randomly allocated from this range for use by their Xen domains.

For a list of IEEE OUI assignments, see `http://standards.ieee.org/regauth/oui/oui.txt`

3.2.2 Booting the Guest Domain

The `xm` tool provides a variety of commands for managing domains. Use the `create` command to start new domains. Assuming you've created a configuration file `myvmconf`

based around `/etc/xen/xmexample2`, to start a domain with virtual machine ID 1 you should type:

```
# xm create -c myvmconf vmid=1
```

The `-c` switch causes `xm` to turn into the domain's console after creation. The `vmid=1` sets the `vmid` variable used in the `myvmconf` file.

You should see the console boot messages from the new domain appearing in the terminal in which you typed the command, culminating in a login prompt.

3.3 Starting / Stopping Domains Automatically

It is possible to have certain domains start automatically at boot time and to have dom0 wait for all running domains to shutdown before it shuts down the system.

To specify a domain is to start at boot-time, place its configuration file (or a link to it) under `/etc/xen/auto/`.

A Sys-V style init script for Red Hat and LSB-compliant systems is provided and will be automatically copied to `/etc/init.d/` during install. You can then enable it in the appropriate way for your distribution.

For instance, on Red Hat:

```
# chkconfig --add xendomains
```

By default, this will start the boot-time domains in runlevels 3, 4 and 5.

You can also use the `service` command to run this script manually, e.g:

```
# service xendomains start
```

Starts all the domains with config files under /etc/xen/auto/.

```
# service xendomains stop
```

Shuts down all running Xen domains.

Part II

Configuration and Management

Chapter 4

Domain Management Tools

This chapter summarizes the management software and tools available.

4.1 Xend

The Xend node control daemon performs system management functions related to virtual machines. It forms a central point of control of virtualized resources, and must be running in order to start and manage virtual machines. Xend must be run as root because it needs access to privileged system management functions.

An initialization script named `/etc/init.d/xend` is provided to start Xend at boot time. Use the tool appropriate (i.e. chkconfig) for your Linux distribution to specify the runlevels at which this script should be executed, or manually create symbolic links in the correct runlevel directories.

Xend can be started on the command line as well, and supports the following set of parameters:

`# xend start`	start xend, if not already running
`# xend stop`	stop xend if already running
`# xend restart`	restart xend if running, otherwise start it
`# xend status`	indicates xend status by its return code

A SysV init script called `xend` is provided to start xend at boot time. `make install` installs this script in `/etc/init.d`. To enable it, you have to make symbolic links in the appropriate runlevel directories or use the `chkconfig` tool, where available. Once xend is running, administration can be done using the `xm` tool.

4.1.1 Logging

As xend runs, events will be logged to `/var/log/xend.log` and (less frequently) to `/var/log/xend-debug.log`. These, along with the standard syslog files, are useful when troubleshooting problems.

4.1.2 Configuring Xend

Xend is written in Python. At startup, it reads its configuration information from the file `/etc/xen/xend-config.sxp`. The Xen installation places an example `xend-config.sxp` file in the `/etc/xen` subdirectory which should work for most installations.

See the example configuration file `xend-debug.sxp` and the section 5 man page `xend-config.sxp` for a full list of parameters and more detailed information. Some of the most important parameters are discussed below.

An HTTP interface and a Unix domain socket API are available to communicate with Xend. This allows remote users to pass commands to the daemon. By default, Xend does not start an HTTP server. It does start a Unix domain socket management server, as the low level utility `xm` requires it. For support of cross-machine migration, Xend can start a relocation server. This support is not enabled by default for security reasons.

Note: the example `xend` configuration file modifies the defaults and starts up Xend as an HTTP server as well as a relocation server.

From the file:

```
#(xend-http-server no)
(xend-http-server yes)
#(xend-unix-server yes)
#(xend-relocation-server no)
(xend-relocation-server yes)
```

Comment or uncomment lines in that file to disable or enable features that you require.

Connections from remote hosts are disabled by default:

```
# Address xend should listen on for HTTP connections, if xend-http-server is
# set.
# Specifying 'localhost' prevents remote connections.
# Specifying the empty string '' (the default) allows all connections.
#(xend-address '')
(xend-address localhost)
```

It is recommended that if migration support is not needed, the `xend-relocation-server` parameter value be changed to "`no`" or commented out.

4.2 Xm

The xm tool is the primary tool for managing Xen from the console. The general format of an xm command line is:

```
# xm command [switches] [arguments] [variables]
```

The available *switches* and *arguments* are dependent on the *command* chosen. The *variables* may be set using declarations of the form `variable=value` and command line declarations override any of the values in the configuration file being used, including the standard variables described above and any custom variables (for instance, the `xmdefconfig` file uses a `vmid` variable).

For online help for the commands available, type:

```
# xm help
```

This will list the most commonly used commands. The full list can be obtained using `xm help --long`. You can also type `xm help` <command> for more information on a given command.

4.2.1 Basic Management Commands

One useful command is `# xm list` which lists all domains running in rows of the following format:

```
name domid memory vcpus state cputime
```

The meaning of each field is as follows:

name The descriptive name of the virtual machine.

domid The number of the domain ID this virtual machine is running in.

memory Memory size in megabytes.

vcpus The number of virtual CPUs this domain has.

state Domain state consists of 5 fields:

- **r** running
- **b** blocked
- **p** paused
- **s** shutdown
- **c** crashed

cputime How much CPU time (in seconds) the domain has used so far.

The `xm list` command also supports a long output format when the `-l` switch is used. This outputs the full details of the running domains in xend's SXP configuration format.

You can get access to the console of a particular domain using the `# xm console` command (e.g. `# xm console myVM`).

Chapter 5

Domain Configuration

The following contains the syntax of the domain configuration files and description of how to further specify networking, driver domain and general scheduling behavior.

5.1 Configuration Files

Xen configuration files contain the following standard variables. Unless otherwise stated, configuration items should be enclosed in quotes: see the configuration scripts in `/etc/xen/` for concrete examples.

kernel Path to the kernel image.

ramdisk Path to a ramdisk image (optional).

memory Memory size in megabytes.

vcpus The number of virtual CPUs.

console Port to export the domain console on (default 9600 + domain ID).

vif Network interface configuration. This may simply contain an empty string for each desired interface, or may override various settings, e.g.

```
vif = [ 'mac=00:16:3E:00:00:11, bridge=xen-br0',
        'bridge=xen-br1' ]
```

to assign a MAC address and bridge to the first interface and assign a different bridge to the second interface, leaving xend to choose the MAC address. The settings that may be overridden in this way are type, mac, bridge, ip, script, backend, and vifname.

disk List of block devices to export to the domain e.g. `disk = [ 'phy:hda1,sda1,r' ]` exports physical device `/dev/hda1` to the domain as `/dev/sda1` with read-only access. Exporting a disk read-write which is currently mounted is dangerous – if you are *certain* you wish to do this, you can specify `w!` as the mode.

dhcp Set to `'dhcp'` if you want to use DHCP to configure networking.

netmask Manually configured IP netmask.

gateway Manually configured IP gateway.

hostname Set the hostname for the virtual machine.

root Specify the root device parameter on the kernel command line.

nfs_server IP address for the NFS server (if any).

nfs_root Path of the root filesystem on the NFS server (if any).

extra Extra string to append to the kernel command line (if any)

Additional fields are documented in the example configuration files (e.g. to configure virtual TPM functionality).

For additional flexibility, it is also possible to include Python scripting commands in configuration files. An example of this is the `xmexample2` file, which uses Python code to handle the `vmid` variable.

5.2 Network Configuration

For many users, the default installation should work "out of the box". More complicated network setups, for instance with multiple Ethernet interfaces and/or existing bridging setups will require some special configuration.

The purpose of this section is to describe the mechanisms provided by xend to allow a flexible configuration for Xen's virtual networking.

5.2.1 Xen virtual network topology

Each domain network interface is connected to a virtual network interface in dom0 by a point to point link (effectively a "virtual crossover cable"). These devices are named `vif`<`domid`>`.`<`vifid`> (e.g. `vif1.0` for the first interface in domain 1, `vif3.1` for the second interface in domain 3).

Traffic on these virtual interfaces is handled in domain 0 using standard Linux mechanisms for bridging, routing, rate limiting, etc. Xend calls on two shell scripts to perform initial configuration of the network and configuration of new virtual interfaces. By default, these scripts configure a single bridge for all the virtual interfaces. Arbitrary routing / bridging configurations can be configured by customizing the scripts, as described in the following section.

5.2.2 Xen networking scripts

Xen's virtual networking is configured by two shell scripts (by default `network-bridge` and `vif-bridge`). These are called automatically by xend when certain events occur, with arguments to the scripts providing further contextual information. These scripts are found by default in `/etc/xen/scripts`. The names and locations of the scripts can be configured in `/etc/xen/xend-config.sxp`.

network-bridge: This script is called whenever xend is started or stopped to respectively initialize or tear down the Xen virtual network. In the default configuration initialization creates the bridge 'xen-br0' and moves eth0 onto that bridge, modifying the routing accordingly. When xend exits, it deletes the Xen bridge and removes eth0, restoring the normal IP and routing configuration.

vif-bridge: This script is called for every domain virtual interface and can configure firewalling rules and add the vif to the appropriate bridge. By default, this adds and removes VIFs on the default Xen bridge.

Other example scripts are available (`network-route` and `vif-route`, `network-nat` and `vif-nat`). For more complex network setups (e.g. where routing is required or integrate with existing bridges) these scripts may be replaced with customized variants for your site's preferred configuration.

5.3 Driver Domain Configuration

5.3.1 PCI

Individual PCI devices can be assigned to a given domain to allow that domain direct access to the PCI hardware. To use this functionality, ensure that the PCI Backend is compiled in to a privileged domain (e.g. domain 0) and that the domains which will be assigned PCI devices have the PCI Frontend compiled in. In XenLinux, the PCI Backend is available under the Xen configuration section while the PCI Frontend is under the architecture-specific "Bus Options" section. You may compile both the backend and the frontend into the same kernel; they will not affect each other.

The PCI devices you wish to assign to unprivileged domains must be "hidden" from your backend domain (usually domain 0) so that it does not load a driver for them. Use the `pciback.hide` kernel parameter which is specified on the kernel command-line and is configurable through GRUB (see Section 2.5). Note that devices are not really hidden from the backend domain. The PCI Backend ensures that no other device driver loads for those devices. PCI devices are identified by hexadecimal slot/funciton numbers (on Linux, use `lspci` to determine slot/funciton numbers of your devices) and can be specified with or without the PCI domain:

(*bus*:*slot*.*func*) example (02:1d.3)

(*domain*:*bus*:*slot*.*func*) example (0000:02:1d.3)

An example kernel command-line which hides two PCI devices might be:

```
root=/dev/sda4 ro console=tty0 pciback.hide=(02:01.f)(0000:04:1d.0)
```

To configure a domU to receive a PCI device:

Command-line: Use the *pci* command-line flag. For multiple devices, use the option multiple times.

```
xm create netcard-dd pci=01:00.0 pci=02:03.0
```

Flat Format configuration file: Specify all of your PCI devices in a python list named *pci*.

```
pci=['01:00.0','02:03.0']
```

SXP Format configuration file: Use a single PCI device section for all of your devices (specify the numbers in hexadecimal with the preceding '0x'). Note that *domain* here refers to the PCI domain, not a virtual machine within Xen.

```
(device (pci
    (dev (domain 0x0)(bus 0x3)(slot 0x1a)(func 0x1)
    (dev (domain 0x0)(bus 0x1)(slot 0x5)(func 0x0)
)
```

There are a number of security concerns associated with PCI Driver Domains that you can read about in Section 9.2.

Chapter 6

Storage and File System Management

Storage can be made available to virtual machines in a number of different ways. This chapter covers some possible configurations.

The most straightforward method is to export a physical block device (a hard drive or partition) from dom0 directly to the guest domain as a virtual block device (VBD).

Storage may also be exported from a filesystem image or a partitioned filesystem image as a *file-backed VBD*.

Finally, standard network storage protocols such as NBD, iSCSI, NFS, etc., can be used to provide storage to virtual machines.

6.1 Exporting Physical Devices as VBDs

One of the simplest configurations is to directly export individual partitions from domain 0 to other domains. To achieve this use the `phy:` specifier in your domain configuration file. For example a line like

```
disk = ['phy:hda3,sda1,w']
```

specifies that the partition `/dev/hda3` in domain 0 should be exported read-write to the new domain as `/dev/sda1`; one could equally well export it as `/dev/hda` or `/dev/sdb5` should one wish.

In addition to local disks and partitions, it is possible to export any device that Linux considers to be "a disk" in the same manner. For example, if you have iSCSI disks or GNBD volumes imported into domain 0 you can export these to other domains using the `phy:` disk syntax. E.g.:

```
disk = ['phy:vg/lvm1,sda2,w']
```

Warning: Block device sharing

Block devices should typically only be shared between domains in a read-only fashion otherwise the Linux kernel's file systems will get very confused as the file system structure may change underneath them (having the same ext3 partition mounted `rw` twice is a sure fire way to cause irreparable damage)! Xend will attempt to prevent you from doing this by checking that the device is not mounted read-write in domain 0, and hasn't already been exported read-write to another domain. If you want read-write sharing, export the directory to other domains via NFS from domain 0 (or use a cluster file system such as GFS or ocfs2).

6.2 Using File-backed VBDs

It is also possible to use a file in Domain 0 as the primary storage for a virtual machine. As well as being convenient, this also has the advantage that the virtual block device will be *sparse* — space will only really be allocated as parts of the file are used. So if a virtual machine uses only half of its disk space then the file really takes up half of the size allocated.

For example, to create a 2GB sparse file-backed virtual block device (actually only consumes 1KB of disk):

```
# dd if=/dev/zero of=vm1disk bs=1k seek=2048k count=1
```

Make a file system in the disk file:

```
# mkfs -t ext3 vm1disk
```

(when the tool asks for confirmation, answer 'y')

Populate the file system e.g. by copying from the current root:

```
# mount -o loop vm1disk /mnt
# cp -ax /{root,dev,var,etc,usr,bin,sbin,lib} /mnt
# mkdir /mnt/{proc,sys,home,tmp}
```

Tailor the file system by editing `/etc/fstab`, `/etc/hostname`, etc. Don't forget to edit the files in the mounted file system, instead of your domain 0 filesystem, e.g. you would edit `/mnt/etc/fstab` instead of `/etc/fstab`. For this example put `/dev/sda1` to root in fstab.

Now unmount (this is important!):

```
# umount /mnt
```

In the configuration file set:

```
disk = ['file:/full/path/to/vm1disk,sda1,w']
```

As the virtual machine writes to its 'disk', the sparse file will be filled in and consume more space up to the original 2GB.

Note that file-backed VBDs may not be appropriate for backing I/O-intensive domains. File-backed VBDs are known to experience substantial slowdowns under heavy I/O workloads, due to the I/O handling by the loopback block device used to support file-backed VBDs in dom0. Better I/O performance can be achieved by using either LVM-backed VBDs (Section 6.3) or physical devices as VBDs (Section 6.1).

Linux supports a maximum of eight file-backed VBDs across all domains by default. This limit can be statically increased by using the *max_loop* module parameter if CONFIG_BLK_DEV_LOOP is compiled as a module in the dom0 kernel, or by using the *max_loop=n* boot option if CONFIG_BLK_DEV_LOOP is compiled directly into the dom0 kernel.

6.3 Using LVM-backed VBDs

A particularly appealing solution is to use LVM volumes as backing for domain file-systems since this allows dynamic growing/shrinking of volumes as well as snapshot and other features.

To initialize a partition to support LVM volumes:

```
# pvcreate /dev/sda10
```

Create a volume group named 'vg' on the physical partition:

```
# vgcreate vg /dev/sda10
```

Create a logical volume of size 4GB named 'myvmdisk1':

```
# lvcreate -L4096M -n myvmdisk1 vg
```

You should now see that you have a `/dev/vg/myvmdisk1` Make a filesystem, mount it and populate it, e.g.:

```
# mkfs -t ext3 /dev/vg/myvmdisk1
# mount /dev/vg/myvmdisk1 /mnt
# cp -ax / /mnt
# umount /mnt
```

Now configure your VM with the following disk configuration:

```
 disk = [ 'phy:vg/myvmdisk1,sda1,w' ]
```

LVM enables you to grow the size of logical volumes, but you'll need to resize the corresponding file system to make use of the new space. Some file systems (e.g. ext3) now support online resize. See the LVM manuals for more details.

You can also use LVM for creating copy-on-write (CoW) clones of LVM volumes (known as writable persistent snapshots in LVM terminology). This facility is new in

Linux 2.6.8, so isn't as stable as one might hope. In particular, using lots of CoW LVM disks consumes a lot of dom0 memory, and error conditions such as running out of disk space are not handled well. Hopefully this will improve in future.

To create two copy-on-write clones of the above file system you would use the following commands:

```
# lvcreate -s -L1024M -n myclonedisk1 /dev/vg/myvmdisk1
# lvcreate -s -L1024M -n myclonedisk2 /dev/vg/myvmdisk1
```

Each of these can grow to have 1GB of differences from the master volume. You can grow the amount of space for storing the differences using the lvextend command, e.g.:

```
# lvextend +100M /dev/vg/myclonedisk1
```

Don't let the 'differences volume' ever fill up otherwise LVM gets rather confused. It may be possible to automate the growing process by using `dmsetup wait` to spot the volume getting full and then issue an `lvextend`.

In principle, it is possible to continue writing to the volume that has been cloned (the changes will not be visible to the clones), but we wouldn't recommend this: have the cloned volume as a 'pristine' file system install that isn't mounted directly by any of the virtual machines.

6.4 Using NFS Root

First, populate a root filesystem in a directory on the server machine. This can be on a distinct physical machine, or simply run within a virtual machine on the same node.

Now configure the NFS server to export this filesystem over the network by adding a line to `/etc/exports`, for instance:

```
/export/vm1root      1.2.3.4/24 (rw,sync,no_root_squash)
```

Finally, configure the domain to use NFS root. In addition to the normal variables, you should make sure to set the following values in the domain's configuration file:

```
root       = '/dev/nfs'
nfs_server = '2.3.4.5'       # substitute IP address of server
nfs_root   = '/path/to/root' # path to root FS on the server
```

The domain will need network access at boot time, so either statically configure an IP address using the config variables `ip`, `netmask`, `gateway`, `hostname`; or enable DHCP (`dhcp='dhcp'`).

Note that the Linux NFS root implementation is known to have stability problems under high load (this is not a Xen-specific problem), so this configuration may not be appropriate for critical servers.

Chapter 7

CPU Management

Xen allows a domain's virtual CPU(s) to be associated with one or more host CPUs. This can be used to allocate real resources among one or more guests, or to make optimal use of processor resources when utilizing dual-core, hyperthreading, or other advanced CPU technologies.

Xen enumerates physical CPUs in a 'depth first' fashion. For a system with both hyperthreading and multiple cores, this would be all the hyperthreads on a given core, then all the cores on a given socket, and then all sockets. I.e. if you had a two socket, dual core, hyperthreaded Xeon the CPU order would be:

socket0				socket1			
core0		core1		core0		core1	
ht0	ht1	ht0	ht1	ht0	ht1	ht0	ht1
#0	#1	#2	#3	#4	#5	#6	#7

Having multiple vcpus belonging to the same domain mapped to the same physical CPU is very likely to lead to poor performance. It's better to use 'vcpus-set' to hot-unplug one of the vcpus and ensure the others are pinned on different CPUs.

If you are running IO intensive tasks, its typically better to dedicate either a hyperthread or whole core to running domain 0, and hence pin other domains so that they can't use CPU 0. If your workload is mostly compute intensive, you may want to pin vcpus such that all physical CPU threads are available for guest domains.

Chapter 8

Migrating Domains

8.1 Domain Save and Restore

The administrator of a Xen system may suspend a virtual machine's current state into a disk file in domain 0, allowing it to be resumed at a later time.

For example you can suspend a domain called "VM1" to disk using the command:

```
# xm save VM1 VM1.chk
```

This will stop the domain named "VM1" and save its current state into a file called `VM1.chk`.

To resume execution of this domain, use the `xm restore` command:

```
# xm restore VM1.chk
```

This will restore the state of the domain and resume its execution. The domain will carry on as before and the console may be reconnected using the `xm console` command, as described earlier.

8.2 Migration and Live Migration

Migration is used to transfer a domain between physical hosts. There are two varieties: regular and live migration. The former moves a virtual machine from one host to another by pausing it, copying its memory contents, and then resuming it on the destination. The latter performs the same logical functionality but without needing to pause the domain for the duration. In general when performing live migration the domain continues its usual activities and—from the user's perspective—the migration should be imperceptible.

To perform a live migration, both hosts must be running Xen / xend and the destination host must have sufficient resources (e.g. memory capacity) to accommodate the

domain after the move. Furthermore we currently require both source and destination machines to be on the same L2 subnet.

Currently, there is no support for providing automatic remote access to filesystems stored on local disk when a domain is migrated. Administrators should choose an appropriate storage solution (i.e. SAN, NAS, etc.) to ensure that domain filesystems are also available on their destination node. GNBD is a good method for exporting a volume from one machine to another. iSCSI can do a similar job, but is more complex to set up.

When a domain migrates, it's MAC and IP address move with it, thus it is only possible to migrate VMs within the same layer-2 network and IP subnet. If the destination node is on a different subnet, the administrator would need to manually configure a suitable etherip or IP tunnel in the domain 0 of the remote node.

A domain may be migrated using the `xm migrate` command. To live migrate a domain to another machine, we would use the command:

```
# xm migrate --live mydomain destination.ournetwork.com
```

Without the `--live` flag, xend simply stops the domain and copies the memory image over to the new node and restarts it. Since domains can have large allocations this can be quite time consuming, even on a Gigabit network. With the `--live` flag xend attempts to keep the domain running while the migration is in progress, resulting in typical down times of just 60–300ms.

For now it will be necessary to reconnect to the domain's console on the new machine using the `xm console` command. If a migrated domain has any open network connections then they will be preserved, so SSH connections do not have this limitation.

Chapter 9

Securing Xen

This chapter describes how to secure a Xen system. It describes a number of scenarios and provides a corresponding set of best practices. It begins with a section devoted to understanding the security implications of a Xen system.

9.1 Xen Security Considerations

When deploying a Xen system, one must be sure to secure the management domain (Domain-0) as much as possible. If the management domain is compromised, all other domains are also vulnerable. The following are a set of best practices for Domain-0:

1. **Run the smallest number of necessary services.** The less things that are present in a management partition, the better. Remember, a service running as root in the management domain has full access to all other domains on the system.
2. **Use a firewall to restrict the traffic to the management domain.** A firewall with default-reject rules will help prevent attacks on the management domain.
3. **Do not allow users to access Domain-0.** The Linux kernel has been known to have local-user root exploits. If you allow normal users to access Domain-0 (even as unprivileged users) you run the risk of a kernel exploit making all of your domains vulnerable.

9.2 Driver Domain Security Considerations

Driver domains address a range of security problems that exist regarding the use of device drivers and hardware. On many operating systems in common use today, device drivers run within the kernel with the same privileges as the kernel. Few or no mechanisms exist to protect the integrity of the kernel from a misbehaving (read ”buggy”) or

malicious device driver. Driver domains exist to aid in isolating a device driver within its own virtual machine where it cannot affect the stability and integrity of other domains. If a driver crashes, the driver domain can be restarted rather than have the entire machine crash (and restart) with it. Drivers written by unknown or untrusted third-parties can be confined to an isolated space. Driver domains thus address a number of security and stability issues with device drivers.

However, due to limitations in current hardware, a number of security concerns remain that need to be considered when setting up driver domains (it should be noted that the following list is not intended to be exhaustive).

1. **Without an IOMMU, a hardware device can DMA to memory regions outside of its controlling domain.** Architectures which do not have an IOMMU (e.g. most x86-based platforms) to restrict DMA usage by hardware are vulnerable. A hardware device which can perform arbitrary memory reads and writes can read/write outside of the memory of its controlling domain. A malicious or misbehaving domain could use a hardware device it controls to send data overwriting memory in another domain or to read arbitrary regions of memory in another domain.

2. **Shared buses are vulnerable to sniffing.** Devices that share a data bus can sniff (and possible spoof) each others' data. Device A that is assigned to Domain A could eavesdrop on data being transmitted by Domain B to Device B and then relay that data back to Domain A.

3. **Devices which share interrupt lines can either prevent the reception of that interrupt by the driver domain or can trigger the interrupt service routine of that guest needlessly.** A devices which shares a level-triggered interrupt (e.g. PCI devices) with another device can raise an interrupt and never clear it. This effectively blocks other devices which share that interrupt line from notifying their controlling driver domains that they need to be serviced. A device which shares an any type of interrupt line can trigger its interrupt continually which forces execution time to be spent (in multiple guests) in the interrupt service routine (potentially denying time to other processes within that guest). System architectures which allow each device to have its own interrupt line (e.g. PCI's Message Signaled Interrupts) are less vulnerable to this denial-of-service problem.

4. **Devices may share the use of I/O memory address space.** Xen can only restrict access to a device's physical I/O resources at a certain granularity. For interrupt lines and I/O port address space, that granularity is very fine (per interrupt line and per I/O port). However, Xen can only restrict access to I/O memory address space on a page size basis. If more than one device shares use of a page in I/O memory address space, the domains to which those devices are assigned will be able to access the I/O memory address space of each other's devices.

9.3 Security Scenarios

9.3.1 The Isolated Management Network

In this scenario, each node has two network cards in the cluster. One network card is connected to the outside world and one network card is a physically isolated management network specifically for Xen instances to use.

As long as all of the management partitions are trusted equally, this is the most secure scenario. No additional configuration is needed other than forcing Xend to bind to the management interface for relocation.

9.3.2 A Subnet Behind a Firewall

In this scenario, each node has only one network card but the entire cluster sits behind a firewall. This firewall should do at least the following:

1. Prevent IP spoofing from outside of the subnet.
2. Prevent access to the relocation port of any of the nodes in the cluster except from within the cluster.

The following iptables rules can be used on each node to prevent migrations to that node from outside the subnet assuming the main firewall does not do this for you:

```
# this command disables all access to the Xen relocation
# port:
iptables -A INPUT -p tcp --destination-port 8002 -j REJECT

# this command enables Xen relocations only from the specific
# subnet:
iptables -I INPUT -p tcp -{}-source 192.168.1.1/8 \
    --destination-port 8002 -j ACCEPT
```

9.3.3 Nodes on an Untrusted Subnet

Migration on an untrusted subnet is not safe in current versions of Xen. It may be possible to perform migrations through a secure tunnel via an VPN or SSH. The only safe option in the absence of a secure tunnel is to disable migration completely. The easiest way to do this is with iptables:

```
# this command disables all access to the Xen relocation port
iptables -A INPUT -p tcp -{}-destination-port 8002 -j REJECT
```

Part III

Reference

Chapter 10

Build and Boot Options

This chapter describes the build- and boot-time options which may be used to tailor your Xen system.

10.1 Top-level Configuration Options

Top-level configuration is achieved by editing one of two files: `Config.mk` and `Makefile`.

The former allows the overall build target architecture to be specified. You will typically not need to modify this unless you are cross-compiling or if you wish to build a PAE-enabled Xen system. Additional configuration options are documented in the `Config.mk` file.

The top-level `Makefile` is chiefly used to customize the set of kernels built. Look for the line:

```
KERNELS ?= linux-2.6-xen0 linux-2.6-xenU
```

Allowable options here are any kernels which have a corresponding build configuration file in the `buildconfigs/` directory.

10.2 Xen Build Options

Xen provides a number of build-time options which should be set as environment variables or passed on make's command-line.

verbose=y Enable debugging messages when Xen detects an unexpected condition. Also enables console output from all domains.

debug=y Enable debug assertions. Implies **verbose=y**. (Primarily useful for tracing bugs in Xen).

debugger=y Enable the in-Xen debugger. This can be used to debug Xen, guest OSes, and applications.

perfc=y Enable performance counters for significant events within Xen. The counts can be reset or displayed on Xen's console via console control keys.

10.3 Xen Boot Options

These options are used to configure Xen's behaviour at runtime. They should be appended to Xen's command line, either manually or by editing `grub.conf`.

noreboot Don't reboot the machine automatically on errors. This is useful to catch debug output if you aren't catching console messages via the serial line.

nosmp Disable SMP support. This option is implied by 'ignorebiostables'.

watchdog Enable NMI watchdog which can report certain failures.

noirqbalance Disable software IRQ balancing and affinity. This can be used on systems such as Dell 1850/2850 that have workarounds in hardware for IRQ-routing issues.

badpage=<page number>,<page number>, ... Specify a list of pages not to be allocated for use because they contain bad bytes. For example, if your memory tester says that byte 0x12345678 is bad, you would place 'badpage=0x12345' on Xen's command line.

com1=<baud>,DPS,<io_base>,<irq> com2=<baud>,DPS,<io_base>,<irq>

Xen supports up to two 16550-compatible serial ports. For example: 'com1=9600, 8n1, 0x408, 5' maps COM1 to a 9600-baud port, 8 data bits, no parity, 1 stop bit, I/O port base 0x408, IRQ 5. If some configuration options are standard (e.g., I/O base and IRQ), then only a prefix of the full configuration string need be specified. If the baud rate is pre-configured (e.g., by the bootloader) then you can specify 'auto' in place of a numeric baud rate.

console=<specifier list> Specify the destination for Xen console I/O. This is a comma-separated list of, for example:

vga Use VGA console and allow keyboard input.

com1 Use serial port com1.

com2H Use serial port com2. Transmitted chars will have the MSB set. Received chars must have MSB set.

com2L Use serial port com2. Transmitted chars will have the MSB cleared. Received chars must have MSB cleared.

The latter two examples allow a single port to be shared by two subsystems

(e.g. console and debugger). Sharing is controlled by MSB of each transmitted/received character. [NB. Default for this option is 'com1,vga']

sync_console Force synchronous console output. This is useful if you system fails unexpectedly before it has sent all available output to the console. In most cases Xen will automatically enter synchronous mode when an exceptional event occurs, but this option provides a manual fallback.

conswitch=<switch-char><auto-switch-char> Specify how to switch serial-console input between Xen and DOM0. The required sequence is CTRL-<switch-char> pressed three times. Specifying the backtick character disables switching. The <auto-switch-char> specifies whether Xen should auto-switch input to DOM0 when it boots — if it is 'x' then auto-switching is disabled. Any other value, or omitting the character, enables auto-switching. [NB. Default switch-char is 'a'.]

nmi=xxx Specify what to do with an NMI parity or I/O error.
'nmi=fatal': Xen prints a diagnostic and then hangs.
'nmi=dom0': Inform DOM0 of the NMI.
'nmi=ignore': Ignore the NMI.

mem=xxx Set the physical RAM address limit. Any RAM appearing beyond this physical address in the memory map will be ignored. This parameter may be specified with a B, K, M or G suffix, representing bytes, kilobytes, megabytes and gigabytes respectively. The default unit, if no suffix is specified, is kilobytes.

dom0_mem=xxx Set the amount of memory to be allocated to domain0. In Xen 3.x the parameter may be specified with a B, K, M or G suffix, representing bytes, kilobytes, megabytes and gigabytes respectively; if no suffix is specified, the parameter defaults to kilobytes. In previous versions of Xen, suffixes were not supported and the value is always interpreted as kilobytes.

tbuf_size=xxx Set the size of the per-cpu trace buffers, in pages (default 1). Note that the trace buffers are only enabled in debug builds. Most users can ignore this feature completely.

sched=xxx Select the CPU scheduler Xen should use. The current possibilities are 'sedf' (default) and 'bvt'.

apic_verbosity=debug,verbose Print more detailed information about local APIC and IOAPIC configuration.

lapic Force use of local APIC even when left disabled by uniprocessor BIOS.

nolapic Ignore local APIC in a uniprocessor system, even if enabled by the BIOS.

apic=bigsmp,default,es7000,summit Specify NUMA platform. This can usually be probed automatically.

In addition, the following options may be specified on the Xen command line. Since domain 0 shares responsibility for booting the platform, Xen will automatically propa-

gate these options to its command line. These options are taken from Linux's command-line syntax with unchanged semantics.

acpi=off,force,strict,ht,noirq,... Modify how Xen (and domain 0) parses the BIOS ACPI tables.

acpi_skip_timer_override Instruct Xen (and domain 0) to ignore timer-interrupt override instructions specified by the BIOS ACPI tables.

noapic Instruct Xen (and domain 0) to ignore any IOAPICs that are present in the system, and instead continue to use the legacy PIC.

10.4 XenLinux Boot Options

In addition to the standard Linux kernel boot options, we support:

xencons=xxx Specify the device node to which the Xen virtual console driver is attached. The following options are supported:

'xencons=off': disable virtual console
'xencons=tty': attach console to /dev/tty1 (tty0 at boot-time)
'xencons=ttyS': attach console to /dev/ttyS0

The default is ttyS for dom0 and tty for all other domains.

Chapter 11

Further Support

If you have questions that are not answered by this manual, the sources of information listed below may be of interest to you. Note that bug reports, suggestions and contributions related to the software (or the documentation) should be sent to the Xen developers' mailing list (address below).

11.1 Other Documentation

For developers interested in porting operating systems to Xen, the *Xen Interface Manual* is distributed in the `docs/` directory of the Xen source distribution.

11.2 Online References

The official Xen web site can be found at:

`http://www.xensource.com`

This contains links to the latest versions of all online documentation, including the latest version of the FAQ.

Information regarding Xen is also available at the Xen Wiki at

`http://wiki.xensource.com/xenwiki/`

The Xen project uses Bugzilla as its bug tracking system. You'll find the Xen Bugzilla at http://bugzilla.xensource.com/bugzilla/.

11.3 Mailing Lists

There are several mailing lists that are used to discuss Xen related topics. The most widely relevant are listed below. An official page of mailing lists and subscription information can be found at

`http://lists.xensource.com/`

xen-devel@lists.xensource.com Used for development discussions and bug reports. Subscribe at:
`http://lists.xensource.com/xen-devel`

xen-users@lists.xensource.com Used for installation and usage discussions and requests for help. Subscribe at:
`http://lists.xensource.com/xen-users`

xen-announce@lists.xensource.com Used for announcements only. Subscribe at:
`http://lists.xensource.com/xen-announce`

xen-changelog@lists.xensource.com Changelog feed from the unstable and 2.0 trees - developer oriented. Subscribe at:
`http://lists.xensource.com/xen-changelog`

Appendix A

Unmodified (VMX) guest domains in Xen with Intel®Virtualization Technology (VT)

Xen supports guest domains running unmodified Guest operating systems using Virtualization Technology (VT) available on recent Intel Processors. More information about the Intel Virtualization Technology implementing Virtual Machine Extensions (VMX) in the processor is available on the Intel website at
`http://www.intel.com/technology/computing/vptech`

A.1 Building Xen with VT support

The following packages need to be installed in order to build Xen with VT support. Some Linux distributions do not provide these packages by default.

Package	Description
dev86	The dev86 package provides an assembler and linker for real mode 80x86 instructions. You need to have this package installed in order to build the BIOS code which runs in (virtual) real mode. If the dev86 package is not available on the x86_64 distribution, you can install the i386 version of it. The dev86 rpm package for various distributions can be found at `http://www.rpmfind.net/linux/rpm2html/search.php?query=dev86&submit=Search`
LibVNCServer	The unmodified guest's VGA display, keyboard, and mouse are virtualized using the vncserver library provided by this package. You can get the sources of libvncserver from `http://sourceforge.net/projects/libvncserver`. Build and install the sources on the build system to get the libvncserver library. The 0.8pre version of libvncserver is currently working well with Xen.
SDL-devel, SDL	Simple DirectMedia Layer (SDL) is another way of virtualizing the unmodified guest console. It provides an X window for the guest console. If the SDL and SDL-devel packages are not installed by default on the build system, they can be obtained from `http://www.rpmfind.net/linux/rpm2html/search.php?query=SDL&submit=Search` , `http://www.rpmfind.net/linux/rpm2html/search.php?query=SDL-devel&submit=Search`

A.2 Configuration file for unmodified VMX guests

The Xen installation includes a sample configuration file, `/etc/xen/xmexample.vmx`. There are comments describing all the options. In addition to the common options that are the same as those for paravirtualized guest configurations, VMX guest configurations have the following settings:

Parameter	Description
kernel	The VMX firmware loader, `/usr/lib/xen/boot/vmxloader`
builder	The domain build function. The VMX domain uses the vmx builder.
acpi	Enable VMX guest ACPI, default=0 (disabled)
apic	Enable VMX guest APIC, default=0 (disabled)
vif	Optionally defines MAC address and/or bridge for the network interfaces. Random MACs are assigned if not given. `type=ioemu` means ioemu is used to virtualize the VMX NIC. If no type is specified, vbd is used, as with paravirtualized guests.

disk	Defines the disk devices you want the domain to have access to, and what you want them accessible as. If using a physical device as the VMX guest's disk, each disk entry is of the form
	`phy:UNAME,ioemu:DEV,MODE,`
	where UNAME is the device, DEV is the device name the domain will see, and MODE is r for read-only, w for read-write. ioemu means the disk will use ioemu to virtualize the VMX disk. If not adding ioemu, it uses vbd like paravirtualized guests.
	If using disk image file, its form should be like
	`file:FILEPATH,ioemu:DEV,MODE`
	If using more than one disk, there should be a comma between each disk entry. For example:
	`disk = ['file:/var/images/image1.img,ioemu:hda,w',`
	`'file:/var/images/image2.img,ioemu:hdb,w']`

cdrom	Disk image for CD-ROM. The default is `/dev/cdrom` for Domain0. Inside the VMX domain, the CD-ROM will available as device `/dev/hdc`. The entry can also point to an ISO file.
boot	Boot from floppy (a), hard disk (c) or CD-ROM (d). For example, to boot from CD-ROM, the entry should be: boot='d'
device_model	The device emulation tool for VMX guests. This parameter should not be changed.
sdl	Enable SDL library for graphics, default = 0 (disabled)
vnc	Enable VNC library for graphics, default = 1 (enabled)
vncviewer	Enable spawning of the vncviewer (only valid when vnc=1), default = 1 (enabled) If vnc=1 and vncviewer=0, user can use vncviewer to manually connect VMX from remote. For example: `vncviewer domain0_IP_address:VMX_domain_id`
ne2000	Enable ne2000, default = 0 (disabled; use pcnet)
serial	Enable redirection of VMX serial output to pty device
localtime	Set the real time clock to local time [default=0, that is, set to UTC].
enable-audio	Enable audio support. This is under development.
full-screen	Start in full screen. This is under development.
nographic	Another way to redirect serial output. If enabled, no 'sdl' or 'vnc' can work. Not recommended.

A.3 Creating virtual disks from scratch

A.3.1 Using physical disks

If you are using a physical disk or physical disk partition, you need to install a Linux OS on the disk first. Then the boot loader should be installed in the correct place. For example `dev/sda` for booting from the whole disk, or `/dev/sda1` for booting from partition 1.

A.3.2 Using disk image files

You need to create a large empty disk image file first; then, you need to install a Linux OS onto it. There are two methods you can choose. One is directly installing it using a VMX guest while booting from the OS installation CD-ROM. The other is copying an installed OS into it. The boot loader will also need to be installed.

To create the image file:

The image size should be big enough to accommodate the entire OS. This example assumes the size is 1G (which is probably too small for most OSes).

```
# dd if=/dev/zero of=hd.img bs=1M count=1 seek=1023
```

To directly install Linux OS into an image file using a VMX guest:

Install Xen and create VMX with the original image file with booting from CD-ROM. Then it is just like a normal Linux OS installation. The VMX configuration file should have these two entries before creating:

```
cdrom='/dev/cdrom' boot='d'
```

If this method does not succeed, you can choose the following method of copying an installed Linux OS into an image file.

To copy a installed OS into an image file:

Directly installing is an easier way to make partitions and install an OS in a disk image file. But if you want to create a specific OS in your disk image, then you will most likely want to use this method.

1. **Install a normal Linux OS on the host machine**
   You can choose any way to install Linux, such as using yum to install Red Hat Linux or YAST to install Novell SuSE Linux. The rest of this example assumes the Linux OS is installed in `/var/guestos/`.

2. **Make the partition table**
   The image file will be treated as hard disk, so you should make the partition table in the image file. For example:

```
# losetup /dev/loop0 hd.img
# fdisk -b 512 -C 4096 -H 16 -S 32 /dev/loop0
press 'n' to add new partition
press 'p' to choose primary partition
press '1' to set partition number
press "Enter" keys to choose default value of "First Cylinder" parameter.
press "Enter" keys to choose default value of "Last Cylinder" parameter.
press 'w' to write partition table and exit
# losetup -d /dev/loop0
```

3. **Make the file system and install grub**

```
# ln -s /dev/loop0 /dev/loop
# losetup /dev/loop0 hd.img
# losetup -o 16384 /dev/loop1 hd.img
# mkfs.ext3 /dev/loop1
# mount /dev/loop1 /mnt
# mkdir -p /mnt/boot/grub
# cp /boot/grub/stage* /boot/grub/e2fs_stage1_5 /mnt/boot/grub
# umount /mnt
# grub
grub> device (hd0) /dev/loop
grub> root (hd0,0)
grub> setup (hd0)
grub> quit
# rm /dev/loop
# losetup -d /dev/loop0
# losetup -d /dev/loop1
```

   The `losetup` option `-o 16384` skips the partition table in the image file. It is the number of sectors times 512. We need `/dev/loop` because grub is expecting a disk device *name*, where *name* represents the entire disk and *name1* represents the first partition.

4. **Copy the OS files to the image**
   If you have Xen installed, you can easily use `lomount` instead of `losetup` and `mount` when coping files to some partitions. `lomount` just needs the partition information.

```
# lomount -t ext3 -diskimage hd.img -partition 1 /mnt/guest
# cp -ax /var/guestos/{root,dev,var,etc,usr,bin,sbin,lib} /mnt/guest
# mkdir /mnt/guest/{proc,sys,home,tmp}
```

5. **Edit the `/etc/fstab` of the guest image**
   The fstab should look like this:

```
# vim /mnt/guest/etc/fstab
/dev/hda1 / ext3 defaults 1 1
none /dev/pts devpts gid=5,mode=620 0 0
none /dev/shm tmpfs defaults 0 0
none /proc proc defaults 0 0
none /sys sysfs efaults 0 0
```

6. **umount the image file**
   `# umount /mnt/guest`

Now, the guest OS image `hd.img` is ready. You can also reference `http://free.oszoo.org` for quickstart images. But make sure to install the boot loader.

A.3.3 Install Windows into an Image File using a VMX guest

In order to install a Windows OS, you should keep `acpi=0` in your VMX configuration file.

A.4 VMX Guests

A.4.1 Editing the Xen VMX config file

Make a copy of the example VMX configuration file `/etc/xen/xmeaxmple.vmx` and edit the line that reads

`disk = [ 'file:/var/images/`*`guest.img`*`,ioemu:hda,w' ]`

replacing *guest.img* with the name of the guest OS image file you just made.

A.4.2 Creating VMX guests

Simply follow the usual method of creating the guest, using the -f parameter and providing the filename of your VMX configuration file:

```
# xend start
# xm create /etc/xen/vmxguest.vmx
```

In the default configuration, VNC is on and SDL is off. Therefore VNC windows will open when VMX guests are created. If you want to use SDL to create VMX guests, set `sdl=1` in your VMX configuration file. You can also turn off VNC by setting `vnc=0`.

A.4.3 Destroy VMX guests

VMX guests can be destroyed in the same way as can paravirtualized guests. We recommend that you type the command

`poweroff`

in the VMX guest's console first to prevent data loss. Then execute the command

`xm destroy` *`vmx_guest_id`*

at the Domain0 console.

A.4.4 VMX window (X or VNC) Hot Key

If you are running in the X environment after creating a VMX guest, an X window is created. There are several hot keys for control of the VMX guest that can be used in the window.

Ctrl+Alt+2 switches from guest VGA window to the control window. Typing `help` shows the control commands help. For example, 'q' is the command to destroy the VMX guest.
Ctrl+Alt+1 switches back to VMX guest's VGA.
Ctrl+Alt+3 switches to serial port output. It captures serial output from the VMX guest. It works only if the VMX guest was configured to use the serial port.

A.4.5 Save/Restore and Migration

VMX guests currently cannot be saved and restored, nor migrated. These features are currently under active development.

Appendix B

Vnets - Domain Virtual Networking

Xen optionally supports virtual networking for domains using *vnets*. These emulate private LANs that domains can use. Domains on the same vnet can be hosted on the same machine or on separate machines, and the vnets remain connected if domains are migrated. Ethernet traffic on a vnet is tunneled inside IP packets on the physical network. A vnet is a virtual network and addressing within it need have no relation to addressing on the underlying physical network. Separate vnets, or vnets and the physical network, can be connected using domains with more than one network interface and enabling IP forwarding or bridging in the usual way.

Vnet support is included in `xm` and xend:

```
# xm vnet-create <config>
```

creates a vnet using the configuration in the file `<config>`. When a vnet is created its configuration is stored by xend and the vnet persists until it is deleted using

```
# xm vnet-delete <vnetid>
```

The vnets xend knows about are listed by

```
# xm vnet-list
```

More vnet management commands are available using the `vn` tool included in the vnet distribution.

The format of a vnet configuration file is

```
(vnet (id       <vnetid>)
      (bridge   <bridge>)
      (vnetif   <vnet interface>)
      (security <level>))
```

White space is not significant. The parameters are:

- `<vnetid>`: vnet id, the 128-bit vnet identifier. This can be given as 8 4-digit hex numbers separated by colons, or in short form as a single 4-digit hex number. The short form is the same as the long form with the first 7 fields zero. Vnet ids must be non-zero and id 1 is reserved.
- `<bridge>`: the name of a bridge interface to create for the vnet. Domains are connected to the vnet by connecting their virtual interfaces to the bridge. Bridge names are limited to 14 characters by the kernel.
- `<vnetif>`: the name of the virtual interface onto the vnet (optional). The interface encapsulates and decapsulates vnet traffic for the network and is attached to the vnet bridge. Interface names are limited to 14 characters by the kernel.
- `<level>`: security level for the vnet (optional). The level may be one of
  - `none`: no security (default). Vnet traffic is in clear on the network.
  - `auth`: authentication. Vnet traffic is authenticated using IPSEC ESP with hmac96.
  - `conf`: confidentiality. Vnet traffic is authenticated and encrypted using IPSEC ESP with hmac96 and AES-128.

  Authentication and confidentiality are experimental and use hard-wired keys at present.

When a vnet is created its configuration is stored by xend and the vnet persists until it is deleted using `xm vnet-delete <vnetid>`. The interfaces and bridges used by vnets are visible in the output of `ifconfig` and `brctl show`.

B.1 Example

If the file `vnet97.sxp` contains

```
(vnet (id 97) (bridge vnet97) (vnetif vnif97)
      (security none))
```

Then `xm vnet-create vnet97.sxp` will define a vnet with id 97 and no security. The bridge for the vnet is called vnet97 and the virtual interface for it is vnif97. To add an interface on a domain to this vnet set its bridge to vnet97 in its configuration. In Python:

```
vif="bridge=vnet97"
```

In sxp:

```
(dev (vif (mac aa:00:00:01:02:03) (bridge vnet97)))
```

Once the domain is started you should see its interface in the output of `brctl show` under the ports for `vnet97`.

To get best performance it is a good idea to reduce the MTU of a domain's interface onto a vnet to 1400. For example using `ifconfig eth0 mtu 1400` or putting `MTU=1400` in `ifcfg-eth0`. You may also have to change or remove cached config files for eth0 under `/etc/sysconfig/networking`. Vnets work anyway, but performance can be reduced by IP fragmentation caused by the vnet encapsulation exceeding the hardware MTU.

B.2 Installing vnet support

Vnets are implemented using a kernel module, which needs to be loaded before they can be used. You can either do this manually before starting xend, using the command `vn insmod`, or configure xend to use the `network-vnet` script in the xend configuration file `/etc/xend/xend-config.sxp`:

```
(network-script        network-vnet)
```

This script insmods the module and calls the `network-bridge` script.

The vnet code is not compiled and installed by default. To compile the code and install on the current system use `make install` in the root of the vnet source tree, `tools/vnet`. It is also possible to install to an installation directory using `make dist`. See the `Makefile` in the source for details.

The vnet module creates vnet interfaces `vnif0002`, `vnif0003` and `vnif0004` by default. You can test that vnets are working by configuring IP addresses on these interfaces and trying to ping them across the network. For example, using machines hostA and hostB:

```
hostA# ifconfig vnif0004 10.0.0.100 up
hostB# ifconfig vnif0004 10.0.0.101 up
hostB# ping 10.0.0.100
```

The vnet implementation uses IP multicast to discover vnet interfaces, so all machines hosting vnets must be reachable by multicast. Network switches are often configured not to forward multicast packets, so this often means that all machines using a vnet must be on the same LAN segment, unless you configure vnet forwarding.

You can test multicast coverage by pinging the vnet multicast address:

```
# ping -b 224.10.0.1
```

You should see replies from all machines with the vnet module running. You can see if vnet packets are being sent or received by dumping traffic on the vnet UDP port:

```
# tcpdump udp port 1798
```

If multicast is not being forwaded between machines you can configure multicast forwarding using vn. Suppose we have machines hostA on 10.10.0.100 and hostB on

10.11.0.100 and that multicast is not forwarded between them. We use vn to configure each machine to forward to the other:

```
hostA# vn peer-add hostB
hostB# vn peer-add hostA
```

Multicast forwarding needs to be used carefully - you must avoid creating forwarding loops. Typically only one machine on a subnet needs to be configured to forward, as it will forward multicasts received from other machines on the subnet.

Appendix C

Glossary of Terms

BVT The BVT scheduler is used to give proportional fair shares of the CPU to domains.

Domain A domain is the execution context that contains a running **virtual machine**. The relationship between virtual machines and domains on Xen is similar to that between programs and processes in an operating system: a virtual machine is a persistent entity that resides on disk (somewhat like a program). When it is loaded for execution, it runs in a domain. Each domain has a **domain ID**.

Domain 0 The first domain to be started on a Xen machine. Domain 0 is responsible for managing the system.

Domain ID A unique identifier for a **domain**, analogous to a process ID in an operating system.

Full virtualization An approach to virtualization which requires no modifications to the hosted operating system, providing the illusion of a complete system of real hardware devices.

Hypervisor An alternative term for **VMM**, used because it means 'beyond supervisor', since it is responsible for managing multiple 'supervisor' kernels.

Live migration A technique for moving a running virtual machine to another physical host, without stopping it or the services running on it.

Paravirtualization An approach to virtualization which requires modifications to the operating system in order to run in a virtual machine. Xen uses paravirtualization but preserves binary compatibility for user space applications.

Shadow pagetables A technique for hiding the layout of machine memory from a virtual machine's operating system. Used in some **VMMs** to provide the illusion of contiguous physical memory, in Xen this is used during **live migration**.

Virtual Block Device Persistant storage available to a virtual machine, providing the

abstraction of an actual block storage device. **VBD**s may be actual block devices, filesystem images, or remote/network storage.

Virtual Machine The environment in which a hosted operating system runs, providing the abstraction of a dedicated machine. A virtual machine may be identical to the underlying hardware (as in **full virtualization**, or it may differ, as in **paravirtualization**).

VMM Virtual Machine Monitor - the software that allows multiple virtual machines to be multiplexed on a single physical machine.

Xen Xen is a paravirtualizing virtual machine monitor, developed primarily by the Systems Research Group at the University of Cambridge Computer Laboratory.

XenLinux A name for the port of the Linux kernel that runs on Xen.