Virtualizing servers with Xen

Evaldo Gardenali

VI International Conference of Unix at UNINET

Outline

- Virtualization
- Xen
- Features
- Scalability
- Performance
- Quality of Service
- Implementation
- Future of Xen

Why?

- Support heterogeneous environments: Linux® 2.4 e 2.6, NetBSD®, Plan9®
  FreeBSD®, OpenSolaris®
- Consolidate work
- Legacy Systems
- Gradual Upgrade
- Service Isolation
- Quality of Service
- Isolated testing and development
- Ease of administration
- Ease of relocation and migration

- Single System Image: Ensim®, Vservers, CKRM, Virtuozzo™, BSD® jail(), Solaris® Zones
  - √ Groups processes in "resource containers"
  - × Hard to get isolation
- Emulation: QEMU, Bochs
  - √ Portable
  - × Extremely slow
- Virtualization: VMware®, VirtualPC®
  - √ Runs unmodified Operating Systems
  - × Virtualizing x86 is inefficient
- User Mode Kernel: User Mode Linux, CoLinux
  - × Guest runs as a process on the host OS
  - × Low performance (I/O, context switches)
- Paravirtualization: Xen®, Denali
  - √ Excellent performance
  - × Requires port to special architecture

- Single System Image: Ensim®, Vservers, CKRM, Virtuozzo™, BSD® jail(), Solaris® Zones
  - √ Groups processes in "resource containers"
  - × Hard to get isolation
- Emulation: QEMU, Bochs
  - √ Portable
  - × Extremely slow
- Virtualization: VMware®, VirtualPC®
  - √ Runs unmodified Operating Systems
  - × Virtualizing x86 is inefficient
- User Mode Kernel: User Mode Linux, CoLinux
  - × Guest runs as a process on the host OS
  - × Low performance (I/O, context switches)
- Paravirtualization: Xen®, Denali
  - √ Excellent performance
  - × Requires port to special architecture

- Single System Image: Ensim®, Vservers, CKRM, Virtuozzo™, BSD® jail(), Solaris® Zones
  - √ Groups processes in "resource containers"
  - × Hard to get isolation
- Emulation: QEMU, Bochs
  - √ Portable
  - × Extremely slow
- Virtualization: VMware®, VirtualPC®
  - √ Runs unmodified Operating Systems
  - × Virtualizing x86 is inefficient
- User Mode Kernel: User Mode Linux, CoLinux
  - × Guest runs as a process on the host OS
  - × Low performance (I/O, context switches)
- Paravirtualization: Xen®, Denali
  - √ Excellent performance
  - × Requires port to special architecture

- Single System Image: Ensim®, Vservers, CKRM, Virtuozzo™, BSD® jail(), Solaris® Zones
  - √ Groups processes in "resource containers"
  - × Hard to get isolation
- Emulation: QEMU, Bochs
  - √ Portable
  - × Extremely slow
- Virtualization: VMware®, VirtualPC®
  - √ Runs unmodified Operating Systems
  - × Virtualizing x86 is inefficient
- User Mode Kernel: User Mode Linux, CoLinux
  - × Guest runs as a process on the host OS
  - × Low performance (I/O, context switches)
- Paravirtualization: Xen®, Denali
  - √ Excellent performance
  - × Requires port to special architecture

- Single System Image: Ensim®, Vservers, CKRM, Virtuozzo™, BSD® jail(), Solaris® Zones
  - √ Groups processes in "resource containers"
  - × Hard to get isolation
- Emulation: QEMU, Bochs
  - √ Portable
  - × Extremely slow
- Virtualization: VMware®, VirtualPC®
  - √ Runs unmodified Operating Systems
  - × Virtualizing x86 is inefficient
- User Mode Kernel: User Mode Linux, CoLinux
  - × Guest runs as a process on the host OS
  - × Low performance (I/O, context switches)
- Paravirtualization: Xen®, Denali
  - √ Excellent performance
  - × Requires port to special architecture

Advantages

For Administrators

- Service Isolation, minimizing damages
- Failure Isolation
- Ease of Administration
- Quality of Service enforcement

For Hosting providers and datacenters

- Offer “Virtual Private Server” services
- Raise Aggregated Value

Advantages

For Administrators

- Service Isolation, minimizing damages
- Failure Isolation
- Ease of Administration
- Quality of Service enforcement

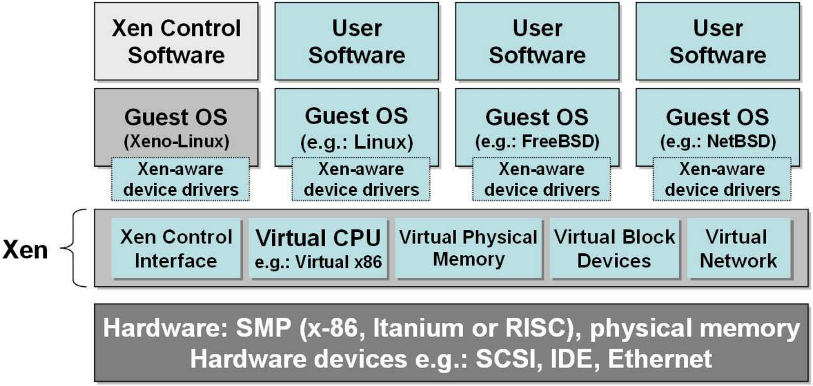
For Hosting providers and datacenters

- Offer "Virtual Private Server" services
- Raise Aggregated Value

Costs!

- Purchase or rent of equipments
- Rack Space
- Colocation costs
- Energy Consumption
- Downtime

Xen architecture



Paravirtualization

- X86 has 4 operation modes (rings)
  - Traditional OSes run on 2 rings: 0 and 3
  - OS/2 uses/used 4 rings
- Hypervisor runs in ring 0
- Operating System kernels: ring 1 ou 2
  - Privileged operations done via hypercalls
  - Needs to be ported to ring 1 or 2
- User processes: ring 3
  - Runs without any modification*

Xen Architecture characteristics

- Kernel runs in ring 1 or 2
- Userland runs unmodified in ring 3
- Privileged operations through hypercalls
- Device access done through hypercalls
- Linux 2.4 Port: less than 3000 lines of code
- Linux 2.6 Port did not modify any “core” files.

Xen 3.0 roadmap

- AGP in Domain 0
- ACPI in Domain 0
- SMP Guests
- Architectures: x86_64, IA64, IBM POWER®
- Intel VT-x (Vanderpool) and AMD Pacifica
- Better management tools
- Network structure optimization

Hardware access in Xen systems

- Domain 0 accesses devices with “native” drivers, through hypercalls
- Domain Us access virtual devices exported by Domain 0
  - Safe asynchronous access through shared memory
  - “Zero-copy” Implementation
  - Network: Use of regular bridging and routing techniques
  - Block Devices: Domain 0 exports any block device (sda4,loop0,vg3,md2,...)
- Access by Privileged Domain Us
  - Native access through hypercalls

Hardware access in Xen systems

- Domain 0 accesses devices with “native” drivers, through hypercalls
- Domain Us access virtual devices exported by Domain 0
  - Safe asynchronous access through shared memory
  - “Zero-copy” Implementation
  - Network: Use of regular bridging and routing techniques
  - Block Devices: Domain 0 exports any block device (sda4,loop0,vg3,md2,...)
- Access by Privileged Domain Us
  - Native access through hypercalls

Hardware access in Xen systems

- Domain 0 accesses devices with “native” drivers, through hypercalls
- Domain Us access virtual devices exported by Domain 0
  - Safe asynchronous access through shared memory
  - “Zero-copy” Implementation
  - Network: Use of regular bridging and routing techniques
  - Block Devices: Domain 0 exports any block device (sda4,loop0,vg3,md2,...)
- Access by Privileged Domain Us
  - Native access through hypercalls

Device Isolation

- "Virtual"
  - Virtual PCI Configuration Space
  - Virtual Interrupts
- Failures don't affect other domains
- It is safe to reboot a domain without affecting others

example

```
root@julia:~# lspci
00:0a.0 Multimedia audio controller: Ensoniq 5880 AudioPCI
root@julia:~#
```

Xen 2.0 Supported Operating Systems

Systems ported to Xen_x86

Operating System	Domain 0	Domain U
Linux® 2.4	$\surd$	$\surd$
Linux® 2.6	$\surd$	$\surd$
NetBSD® 3.0	$\surd$	$\surd$
Plan9®		$\surd$
FreeBSD®		$\surd$
OpenSolaris®		$\surd$
Windows®		$\surd$

Dynamic memory management

- Idle Linux® domain can consume as low as 4MB RAM
- Maximum memory footprint configurable at run-time from Domain 0
- Better use of memory, avoiding Swap
- Control under Domain 0 (xm) or Domain U (/proc/xen/balloon)
- Balloon Auto-Control

Pause: temporary interruption

- Interrupts domain execution
- Stays ready to resume
- xm pause domínio
- xm unpause domínio

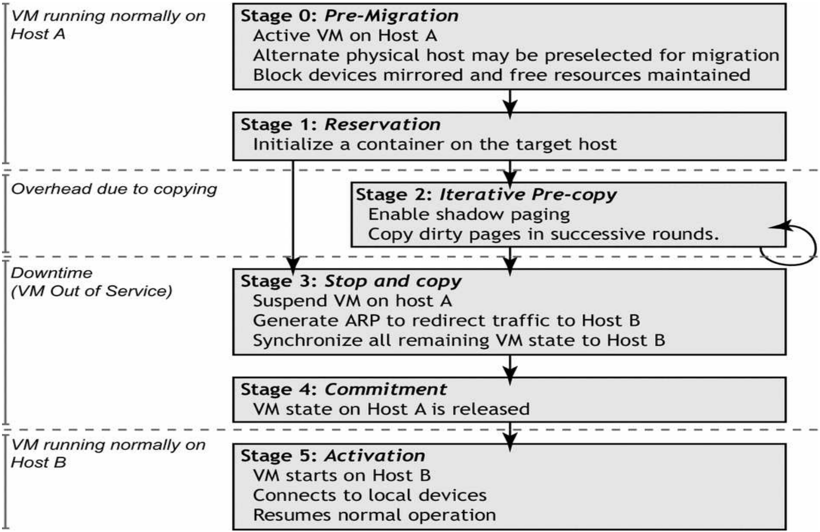
Save: domain suspension

- Interrupts domain execution
- Saves machine state (RAM, registers) to a file
- Destroys the running domain
- Can be used when upgrading domain0
- xm save domínio arquivo
- xm restore arquivo

Live Migration

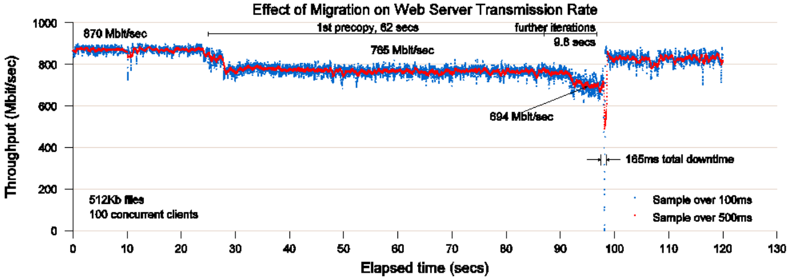
- Transfers a running OS to another host
- “Downtime” of a few milliseconds!
- Obeys bandwidth limits
- Needs shared devices between source and target machines
- Simple!
  - xm migrate –live –resource 70 DominioA OutroHostXen
  - 70Mbit Limit

Live Migration: How It Works



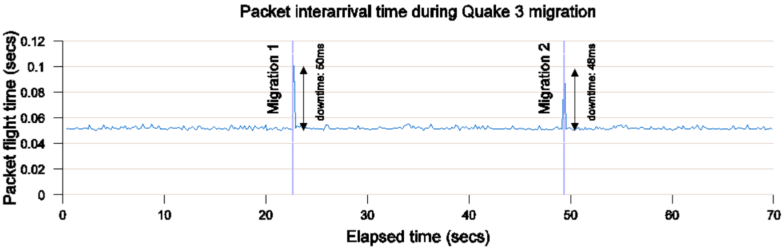
Live Migration: HTTPd

- 512kb files, 100 concurrent clients
- Downtime: 165 ms!



Live Migration: Quake 3

- 6 clients, 64MB
- Total transferred: 88MB (1.37x)
- Downtimes: 50 ms and 48 ms

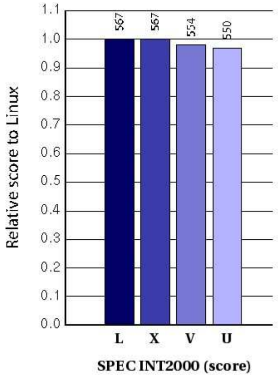


Scalability

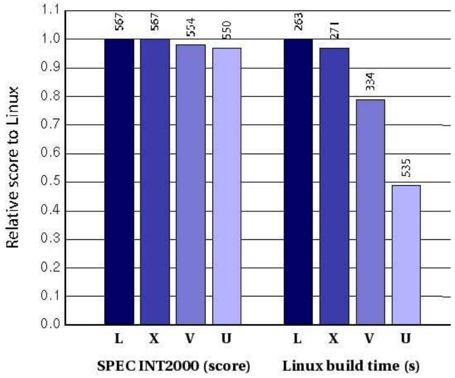
- Memory Overhead per domain: 20kb
- Minimal CPU time Overhead
- Practical limit: memory!
- PC scales well up to 100 domains approximately[1]

[1] Depending on the allocated workload

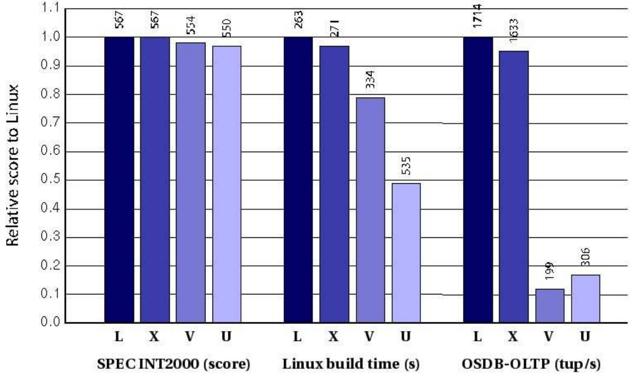
SPECint2000



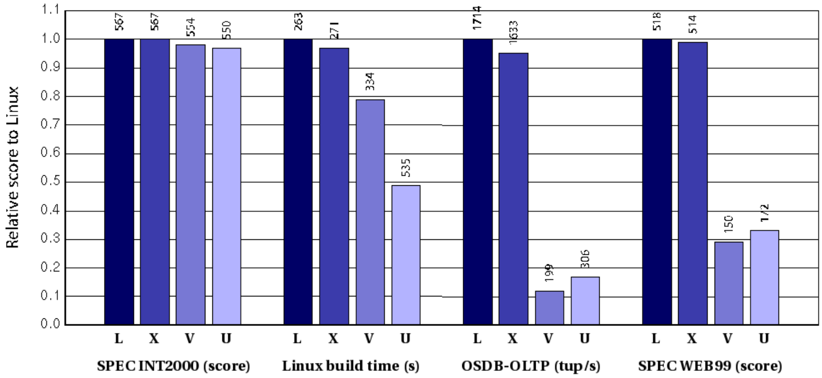
Linux build


Relative score to Linux
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
567
567
554
550
263
271
334
535
L X V U
L X V U
SPEC INT2000 (score)
Linux build time (s)

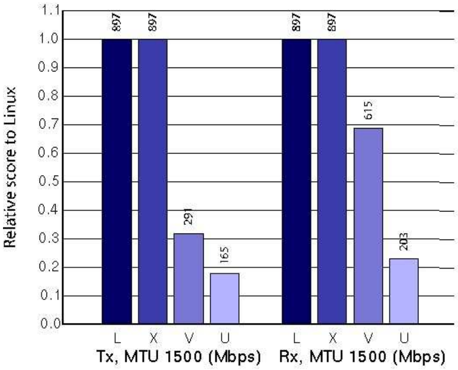
Database transactions



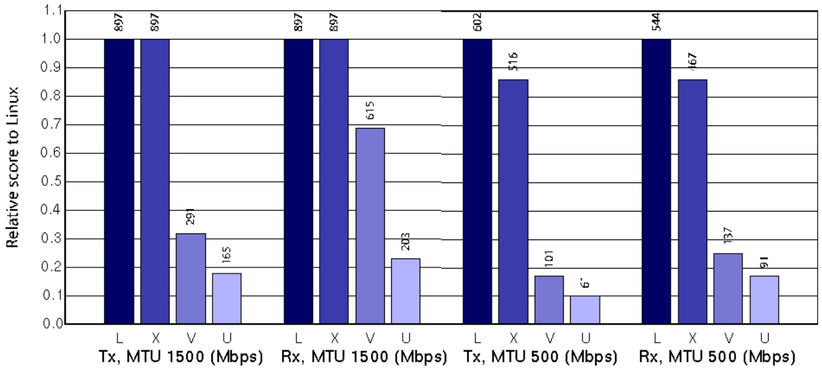
Web Requests



MTU 1500: bulk transfer



MTU 500: Interactive content



Quality of Service Management

- Manage CPU utilization
- Flexible
- Schedulers
  - Round-Robin
  - Borrowed Virtual Time
  - Atropos*
  - Fair Borrowed Virtual Time
  - Simple-Earliest Deadline First

Round-Robin

- Simple sequential scheduler
- Must not be used in production!
- Selected "sched=rrobin"
- Global Parameter

rr_slice Timeslice for each domain

BVT: Borrowed Virtual Time

- BVT provides proportionally fair time slices for each domain
- Experience: Heavy I/O gets penalized
  Compensated by the use of "warp"
- Default scheduler, selected with "sched=bvt"

BVT: Configuration

- Global Parameters
  - ctx_allow: Context Switch Allowance
    Minimum time to run before a domain can be preempted
- Domain Parameters

  mcuadv Minimum Charge Unit Advance, inverse of the CPU weight

  warpback Boolean, allows warping of domains, reducing latency

  warp "Virtual Time" quantity a domain is able to subtract

  warpl Maximum time a domain can run warped, 0 = no limit

  warpu Minimum time to run unwarped before warping again

Atropos

- Soft Real Time
- Selected with “sched=atropos”
- Domain Parameters

period	Regular guaranteed period
slice	Guaranteed timeslice each period cycle
latency	Domain re-scheduling latency
xtratime	Boolean: Can extra time be allocated?

Overview: Implementation

1. Storage Strategy definition
2. Install Operating System
3. Install Xen Hypervisor
4. Install userland tools
5. Prepare Domain 0 kernel
6. Network Configuration
7. Virtual Machine Configuration
8. Install Virtual Machine

Installing Xen Hypervisor

- Build Xen - optional
- Prepare custom Domain 0 kernel - optional
- Install GRUB
- Copy xen.gz and kernel to /boot
- Configure GRUB

Configuring GRUB

/boot/grub/menu.lst - Linux

```
title Xen
root (hd0,1)
kernel /boot/xen.gz dom0_mem=65536
module /boot/vmlinuz-xen0 root=/dev/sda4 ro console=tty0
```

/grub/menu.lst - NetBSD

```
title Xen
root (hd0,0,a)
kernel /xen.gz dom0_mem=65536
module /netbsd
```

Installing userland tools

Dependencies

- iproute2
- bridge-utils (brctl)
- Python
- Twisted (make install-twisted on source directory)
- Compiler toolchain
- libcurl
- zlib
- LaTeX and transfig for the documentation

Installing userland tools

Installing on Linux® - tarball

cd xen-2.0-install
sh ./install.sh
Add “xend start” to your init scripts
Note: Most distributions already have Xen packages

Installing on NetBSD®

cd /usr/pkgsrc/sysutils/xentools20
make install
echo xend=YES >> /etc/rc.conf

Preparing custom Linux® kernel

- Regular linux configuration routine
- Linux needs Xen patches included on the source tarball

Configuring and Building Linux

From Xen source directory:

```
# cd linux-2.6.xx
# make ARCH=xen menuconfig
# cd ..
# make
```

Preparing custom NetBSD® kernel

- Standard configuration and build procedure
- Does not need external patches

Configuring and building kernel

```
# cd /usr/src/sys/arch/i386/conf
# cp XEN0 MYXEN0
# vi MYXEN0
# cd /usr/src
# ./build.sh kernel=MYXEN0
```

Domain Configuration

Example: /etc/xen/example

```
kernel = "/boot/linux-2.6-xenU"
memory = 64
name = example
cpu = -1
nics = 1
cpuweight = 0.1
vif = [ 'mac=01:23:45:67:89:AB, bridge=xen-br0' ]
disk = [ 'file:/path/test-hda1,hda1,w',
         'file:/path/test-hda2,hda2,w' ]
root = "/dev/hda2 ro"
extra = ""
autorestart = True
```

Installing a Linux Domain

- XenU installer
- Bootstrap tools (ex: debootstrap, rpmstrap, yum)
- QEMU
- Tarballs
- ROOT=/mnt/dominio installpkg
  /mnt/cdrom/slackware/{a,ap,n}/*tgz

Installing a NetBSD domain

- kernel = "/boot/netbsd-INSTALL_XENU"
- Packages source:
  - CD: 'phy:/dev/cdrom,cd0d,r', device xbd1d
  - ISO: 'file:/home/foo/i386cd.iso,cd0d,r' device xbd1d
  - Rede: Define networking parameters
- Normal NetBSD install (sysinst)
- Disable virtual terminals

Defining QoS

- Define Scheduler
- Define Global Parameters
- Define individual parameters for each domain

Scheduler

sched=bvt (default)

Parameters

xm bvt_ctxallow *ctxallow*
xm bvt *domínio mcuadv warpback warpvalue warpl warpu*
xm bvt *domínioB 20 0 0 0 0*
xm bvt *domínioB 10 0 0 0 0*

Defining QoS

- Define Scheduler
- Define Global Parameters
- Define individual parameters for each domain

Scheduler

sched=bvt (default)

Parameters

xm bvt_ctxallow *ctxallow*
xm bvt *domínio mcuadv warpback warpvalue warpl warpu*
xm bvt *domínioB 20 0 0 0 0*
xm bvt *domínioB 10 0 0 0 0*

Defining QoS

- Define Scheduler
- Define Global Parameters
- Define individual parameters for each domain

Scheduler

sched=bvt (default)

Parameters

\# xm bvt_ctxallow *ctxallow*

\# xm bvt *domínio mcuadv warpback warpvalue warpl warpu*

\# xm bvt *domínioB 20 0 0 0 0*

\# xm bvt *domínioB 10 0 0 0 0*

Delegating hardware to domains

- Hide the device from Domain 0
- Declare device on Domain U configuration
- Use a domU kernel with support for PCI and your device

/grub/menu.lst

```
kernel /xen.gz dom0_mem=65536 physdev_dom0_hide=(00:0a.0)
```

/etc/xen/test

```
pci = [ '00,0a,00' ]
```

Delegating hardware to domains

- Hide the device from Domain 0
- Declare device on Domain U configuration
- Use a domU kernel with support for PCI and your device

/grub/menu.lst

```
kernel /xen.gz dom0_mem=65536 physdev_dom0_hide=(00:0a.0)
```

/etc/xen/test

```
pci = [ '00,0a,00' ]
```

Delegating hardware to domains

- Hide the device from Domain 0
- Declare device on Domain U configuration
- Use a domU kernel with support for PCI and your device

/grub/menu.lst

```
kernel /xen.gz dom0_mem=65536 physdev_dom0_hide=(00:0a.0)
```

/etc/xen/test

```
pci = [ '00,0a,00' ]
```

Back-End domains

- Device "Servers"
  - Block Devices
  - Network Devices

Enabling Back-end Feature

netif=yes
blkif=yes

Using devices from other Back-Ends

disk = ['file:/path/test-hda1,hda1,w,dom3']
vif = ['mac=00:11:22:33:44:55:66,
bridge=xen-br3, backend=dom5']

Back-End domains

- Device "Servers"
  - Block Devices
  - Network Devices

Enabling Back-end Feature

netif=yes
blkif=yes

Using devices from other Back-Ends

disk = ['file:/path/test-hda1,hda1,w,dom3']
vif = ['mac=00:11:22:33:44:55:66,
bridge=xen-br3, backend=dom5']

Back-End domains

- Device "Servers"
  - Block Devices
  - Network Devices

Enabling Back-end Feature

netif=yes
blkif=yes

Using devices from other Back-Ends

disk = ['file:/path/test-hda1,hda1,w,dom3']
vif = ['mac=00:11:22:33:44:55:66,
bridge=xen-br3, backend=dom5']

Roadmap

- Balloon Auto Control
- Load Balancing
- Node Evacuation
- Storage Subsystem
- Internet Suspend Resume
- Fault Tolerance
- VM fork
- Secure Virtualization

Some References

- http://www.cl.cam.ac.uk/Research/SRG/netos/xen
- http://www.xensource.com/
- http://netbsd.org/Ports/xen/
- http://www.opensolaris.org/os/community/xen/
- http://www.freesoftwaremagazine.com/free_issues/issue_05/focus-xen/
- http://www.kernelthread.com/publications/virtualization/
- http://www.fedoraproject.org/wiki/FedoraXenQuickstart
- http://citeseer.ist.psu.edu/407687.html

?

Evaldo Gardenali
evaldo@gardenali.biz

Feito com $\LaTeX 2_\epsilon$