Xen and the Art of Virtualization

UCL Computer Science

CS M038 / GZ06

Nikola Gvozdiev
Georgian Mihaila
Outline

Xen and the Art of Virtualization – Ian Pratt et al.

I. The Art of Virtualization
II. Xen, goals and design
III. Xen evaluation
IV. The future looks bright
Typical system components

Operating System

Hardware
Basic idea

- Linux
- Mac OS X
- Windows XP

Virtualization layer

Hardware
Some definitions

• **Host/guest**: machine or software

• **Domain**: running VM + guest OS executes

• **Hypervisor** or Virtual Machine Monitor (**VMM**): software or firmware that creates a virtual machine on the host hardware
Why virtualize?

• More resources, subdivide for better use

• 1+ app(s)/OS/server:
  # servers  sys admin complexity & time

  • Scarce resources & unwilling cooperation: resource containers & accountability
Some applications

• Resource optimization

• Infrastructure as a Service (IaaS)

• Application mobility/migration

• Containers/virus/worm ( sandboxing)
Types of hardware virtualization

(software, memory, storage, data, network)

• **Full virtualization** (VMware)
  – run OS/software unmodified

• **Partial virtualization**
  – software may need modification to run

• **Paravirtualization** (Xen)
  – software unmodified runs in modified OS as separate system
Outline

Xen and the Art of Virtualization – Ian Pratt et al.

I. The Art of Virtualization
II. Xen, goals and design
III. Xen evaluation
IV. The future looks bright
Virtualization, at what cost?

- Specialized hardware $ 
- No commodity OS ♥ 
- 100% binary compatibility, speed 
- Speed $\Rightarrow$ security ☠ or functionality 

- Resource isolation/performance guarantees? 
- So far, best effort provisioning $\Rightarrow$ risk DoS
Xen goals

- x86 architecture => $+=0
- Commodity OS (Linux, BSD, XP) => ♥++
- Share resources safely & in a managed fashion => ☠ -- ♥++
- No performance/functionality sacrifice => ♥++
- Up to 100 VM instances on the same server => ?++
- (Free -GNU/GPL- => $--)
The Xen paper

Focus on the VMM design
- How to multiplex physical resources at the granularity of an entire OS and provide performance isolation between guest OSes?

Prototype
- Full Linux, partial Win XP, ongoing NetBSD port
### The Virtual Machine Interface (VMI)

<table>
<thead>
<tr>
<th>Memory Management</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Segmentation</td>
<td>Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.</td>
</tr>
<tr>
<td>Paging</td>
<td>Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated discontiguous machine pages.</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td></td>
</tr>
<tr>
<td>Protection</td>
<td>Guest OS must run at a lower privilege level than Xen.</td>
</tr>
<tr>
<td>Exceptions</td>
<td>Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.</td>
</tr>
<tr>
<td>System Calls</td>
<td>Guest OS may install a ‘fast’ handler for system calls, allowing direct calls from an application into its guest OS and avoiding indirection through Xen on every call.</td>
</tr>
<tr>
<td>Interrupts</td>
<td>Hardware interrupts are replaced with a lightweight event system.</td>
</tr>
<tr>
<td>Time</td>
<td>Each guest OS has a timer interface and is aware of both ‘real’ and ‘virtual’ time.</td>
</tr>
<tr>
<td><strong>Device I/O</strong></td>
<td></td>
</tr>
<tr>
<td>Network, Disk, etc.</td>
<td>Virtual devices are elegant and simple to access. Data is transferred using asynchronous I/O rings. An event mechanism replaces hardware interrupts for notifications.</td>
</tr>
</tbody>
</table>

**Table 1: The paravirtualized x86 interface.**
Typical System call

Program executing

Program blocked

call `read()`

Kernel performing read

`read()` returns

Program executing
Virtualizing CPU

OS is most privileged hypervisor

x86

Applications

guest OS

Hypervisor
Control transfer

Domain (guest OS)
- Hypercall interface (synchronous software trap)

Xen (VMM)
- Event mechanism (asynchronous)
High level view

User mode (low privilege)

Kernel

User program

Sys call (set instructions)

Kernel

CPU/Driver

Return - resume

Put data from registers to where it is needed

Write device registers to initiate I/O

Translate sys call to procedure call (set instructions)

*Loaded at boot time
High level view

User program

Return - resume

Sys call (set instructions)

Code executing in Ring 0 silently fails

User mode (low privilege)

Kernel mode (high privilege)

Put data from registers to where it is needed

Write device registers to initiate I/O

Translate sys call to procedure call (set instructions)

*Loaded at boot time
Exceptions

• Typical exceptions

  – **System calls**: register ‘fast’ exception handler (in Exception Vector Table) which is accessed directly by the processor, without indireciting via ring 0

  – **Page faults**: code executing in ring 0 only can read the faulting address from register CR2, therefore faults are delivered via Xen
Virtualizing CPU cont’d

• CPU scheduling
  – Borrowed Virtual Time (BVT) scheduling algorithm

Work-conserving
Low-latency wake-up (dispatch)

Timers: real, virtual, wall-clock
Memory mapping (relocation) cont’d

Shared Kernel Memory

Virtual Memory Addresses

Process 1

Process 2

Process 3

0xFFFF FFFF

0xC000 0000

0xBFFF FFFF

0x0000 0000

1GB

3GB
Virtualizing memory management (cont’d)

- x86 architecture
  - **No** software managed TLB (walk the page table structure in hardware to recover misses) and no tags (flush TLB for address space switch)

- Guest OSes are responsible for allocating and managing the hardware page tables, minimizing Xen involvement

- Xen exists in a 64MB section at the top of every address space, avoiding a TLB flush when entering and leaving the hypervisor
Xen

I need a new page table

Guest OS

Pool of reserved memory
Xen

Guest OS

I’ll get it myself

Pool of reserved memory
Xen

Guest OS

Register

Pool of reserved memory

Xen
Xen

Guest OS

Pool of reserved memory

Read

Update

Write (batch, up to 8 MB)
Xen

- Reduce memory footprint
- Avoid copies
Virtualizing Device I/O
(disk, network interface)

• Xen exposes device abstractions

• I/O data is transferred to and from each domain via Xen, using shared memory, asynchronous buffer descriptor rings
Data Transfer: I/O Rings

Figure 2: The structure of asynchronous I/O rings, which are used for data transfer between Xen and guest OSes.
Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including Domain0 running control software in a XenoLinux environment.
Outline

Xen and the Art of Virtualization – Ian Pratt et al.

I. The Art of Virtualization
II. Xen, goals and design
III. Xen evaluation
IV. The future looks bright
Cost of porting an OS

<table>
<thead>
<tr>
<th>OS subsection</th>
<th># lines</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linux</td>
</tr>
<tr>
<td>Architecture-independent</td>
<td>78</td>
</tr>
<tr>
<td>Virtual network driver</td>
<td>484</td>
</tr>
<tr>
<td>Virtual block-device driver</td>
<td>1070</td>
</tr>
<tr>
<td>Xen-specific (non-driver)</td>
<td>1363</td>
</tr>
<tr>
<td>Total</td>
<td>2995</td>
</tr>
<tr>
<td>(Portion of total x86 code base)</td>
<td>1.36%</td>
</tr>
</tbody>
</table>

Table 2: The simplicity of porting commodity OSes to Xen. The cost metric is the number of lines of reasonably commented and formatted code which are modified or added compared with the original x86 code base (excluding device drivers).

In both OSes, the architecture-specific sections are effectively a port of the x86 code to their paravirtualized architecture.
Evaluation

• Based on Linux 2.4.21 (neither XP nor NetBSD fully functional)

• Thoroughly compared to 2 other systems
  – VMware Workstation (binary translation)
  – UML (run Linux as a Linux process)

• Performs better than solutions with restrictive licenses (ESX Server)
Relative Performance

CPU-bound tasks

- Relatively easy for all VMMs
- Little interaction with the OS

Legend

L – Linux
X – XenoLinux
V – VMware
U – User-mode Linux
Relative Performance

Tasks with more I/O

- About 7% of the time spent in OS doing I/O and memory management
- This portion of time gets expanded for each VMM but to a different extent
Relative Performance
As expected *fork*, *exec* and *sh* require large number of page updates which slow things down.

On the up side these can be batched (up to 8MB of address space constructed per hypercall)
## Microbenchmarks

<table>
<thead>
<tr>
<th>Config</th>
<th>null call</th>
<th>null I/O</th>
<th>openslct stat</th>
<th>sig closeTCP</th>
<th>sig inst</th>
<th>fork hndl</th>
<th>exec proc</th>
<th>proc proc</th>
<th>proc proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-SMP</td>
<td>0.53</td>
<td>0.81</td>
<td>2.10</td>
<td>3.51</td>
<td>23.2</td>
<td>0.83</td>
<td>2.94</td>
<td>143</td>
<td>601</td>
</tr>
<tr>
<td>L-UP</td>
<td>0.45</td>
<td>0.50</td>
<td>1.28</td>
<td>1.92</td>
<td>5.70</td>
<td>0.68</td>
<td>2.49</td>
<td>110</td>
<td>530</td>
</tr>
<tr>
<td>Xen</td>
<td>0.46</td>
<td>0.50</td>
<td>1.22</td>
<td>1.88</td>
<td>5.69</td>
<td>0.69</td>
<td>1.75</td>
<td>198</td>
<td>768</td>
</tr>
<tr>
<td>VMW</td>
<td>0.73</td>
<td>0.83</td>
<td>1.88</td>
<td>2.99</td>
<td>11.1</td>
<td>1.02</td>
<td>4.63</td>
<td>874</td>
<td>2k3</td>
</tr>
<tr>
<td>UML</td>
<td>24.7</td>
<td>25.1</td>
<td>36.1</td>
<td>62.8</td>
<td>39.9</td>
<td>26.0</td>
<td>46.0</td>
<td>21k</td>
<td>33k</td>
</tr>
</tbody>
</table>

Table 3: `lmbench`: Processes - times in $\mu s$

- Hmmm ... no calls into XEN yet ...
Microbenchmarks

- Overhead due to a hypercall when switching context in a guest OS (in order to change base of page table)
- The larger the working set the smaller the relative overhead

Table 4: lmbench: Context switching times in $\mu$s

<table>
<thead>
<tr>
<th>Config</th>
<th>2p 0K</th>
<th>2p 16K</th>
<th>2p 64K</th>
<th>8p 16K</th>
<th>8p 64K</th>
<th>16p 16K</th>
<th>16p 64K</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-SMP</td>
<td>1.69</td>
<td>1.88</td>
<td>2.03</td>
<td>2.36</td>
<td>26.8</td>
<td>4.79</td>
<td>38.4</td>
</tr>
<tr>
<td>L-UP</td>
<td>0.77</td>
<td>0.91</td>
<td>1.06</td>
<td>1.03</td>
<td>24.3</td>
<td>3.61</td>
<td>37.6</td>
</tr>
<tr>
<td>Xen</td>
<td>1.97</td>
<td>2.22</td>
<td>2.67</td>
<td>3.07</td>
<td>28.7</td>
<td>7.08</td>
<td>39.4</td>
</tr>
<tr>
<td>VMW</td>
<td>18.1</td>
<td>17.6</td>
<td>21.3</td>
<td>22.4</td>
<td>51.6</td>
<td>41.7</td>
<td>72.2</td>
</tr>
<tr>
<td>UML</td>
<td>15.5</td>
<td>14.6</td>
<td>14.4</td>
<td>16.3</td>
<td>36.8</td>
<td>23.6</td>
<td>52.0</td>
</tr>
</tbody>
</table>
Microbenchmarks

- 2 transitions into XEN
  - One for the page fault handler
  - One to actually get the page

Table 5: lmbench: File & VM system latencies in $\mu$s
### Microbenchmarks

<table>
<thead>
<tr>
<th></th>
<th>TCP MTU 1500 TX</th>
<th>TCP MTU 1500 RX</th>
<th>TCP MTU 500 TX</th>
<th>TCP MTU 500 RX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>897</td>
<td>897</td>
<td>602</td>
<td>544</td>
</tr>
<tr>
<td>Xen</td>
<td>897 (-0%)</td>
<td>897 (-0%)</td>
<td>516 (-14%)</td>
<td>467 (-14%)</td>
</tr>
<tr>
<td>VMVW</td>
<td>291 (-68%)</td>
<td>615 (-31%)</td>
<td>101 (-83%)</td>
<td>137 (-75%)</td>
</tr>
<tr>
<td>UML</td>
<td>165 (-82%)</td>
<td>203 (-77%)</td>
<td>61.1 (-90%)</td>
<td>91.4 (-83%)</td>
</tr>
</tbody>
</table>

*Table 6: ttcp: Bandwidth in Mb/s*

- Page flipping really pays off – no unnecessary data copying
- More overhead for smaller packets – we still need to deal with every header
Concurrent VMs

- Unexpectedly low SMP performance for 1 instance of Apache
- As expected adding another domain leads to a sharp jump in performance under XEN
- More domains – more overhead

Figure 4: SPEC WEB99 for 1, 2, 4, 8 and 16 concurrent Apache servers: higher values are better.
Concurrent VMs

- Performance differentiation works as expected with IR
- But fails with OLTP
- Probably due to inefficiencies with the disk scheduling algorithm
- Bands matter !!!
Isolation

- Run uncooperative user applications, see if they bring down the system
- 2 “bad” domains vs 2 “good” ones
- XEN delivers good performance even in this case
- What about an uncooperative OS?
Scalability

- Very low footprint per domain (4 - 6MB memory, 20KB state)
- Benchmark is compute-bound and Linux assigns long time slices, XEN needs some tweaking
- Even without it does pretty well (but no absolute values)
Criticism

• No comparison between fundamentally similar techniques (e.g. the big IBM mainframe)
• Memory footprint almost not mentioned
• Most tests performed with limited number of domains, while the paper’s main goal is to demonstrate performance with 100 domains
• Benchmarks used relevant today?
Related work

- Started by IBM in the 70s
- Resurgence lately (at time of publication) as hardware got fast enough to run multiple OS
- Unlike in other solutions isolation/security is “implicitly” enforced by the hypervisor
- One of the first attempts at paravirtualization the other two are:
  - IBM zSeries mainframes (Linux)
  - Denali (quite a bit different)
# Xen vs. Denali?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Xen</th>
<th>Denali</th>
</tr>
</thead>
<tbody>
<tr>
<td># VM</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>Target existing ABI</td>
<td>Yes</td>
<td>No (does not fully support x86 segmentation, used in Net BSD, Linux, Windows XP)</td>
</tr>
<tr>
<td>Supports application multiplexing</td>
<td>Yes</td>
<td>No (1 app/OS)</td>
</tr>
<tr>
<td>Supports multiple address spaces</td>
<td>Yes</td>
<td>No (prototype virtual MMU developed, helping Denali) =&gt; no evaluation</td>
</tr>
<tr>
<td>Memory management support at the virtualization layer</td>
<td>Yes</td>
<td>No (VMM performs all paging to and from disk)</td>
</tr>
<tr>
<td>Performance isolation</td>
<td>Yes, each guest OS performs own paging using guaranteed memory reservation and disk allocation</td>
<td>No, malicious VM can encourage thrashing behavior</td>
</tr>
<tr>
<td>Virtualizes namespaces of all machine resources</td>
<td>No (Secure access control within hypervisor only)</td>
<td>Yes (resource isolation through impossible naming)</td>
</tr>
</tbody>
</table>
Future work and impact

- Huge impact of the paper, sparked a lot of interest, the project is still very much alive
- They did actually complete the XP port, but due to licensing restrictions it never got published
- Currently only supports Windows as guest if the hardware supports virtualization
- More recent versions of the project try to push complexity away from the hypervisor
- However, paravirtualization nowadays is only used if the hardware does not support virtualization natively
Any questions?

There's no need to worry about the server virtualization project.

In Phase One, a team of blind monkeys will unplug unnecessary servers.

In Phase Two, the monkeys will hurl software at whatever is left.

Voilà!