ParaVirtualization

Xiaohan Fu, Xinyun Lu, Tianrui Chen
Contents

● Introduction of Paravirtualization
● CPU Virtualization
● Memory Virtualization
● I/O Virtualization
● Summary Discussion
Introduction
Recap - Virtualization Principles

Popek and Goldberg’s virtualization principles in 1974:

- Fidelity
- Performance
- Safety
Recap - Privileged and Sensitive Instructions

Goldberg (1974) two classes of instructions

– privileged instructions: those that trap when in user mode

– sensitive instructions: those that modify or depends on hardware configs
Recap - Possible Virtualization Implementation

Complete Machine Emulation

- VMM implements the complete hardware architecture in software
- VMM steps through VM’s every instruction and update emulated hardware as needed

Pros: easy to handle, complete isolation

Cons: super slow
Recap - Possible Virtualization Implementation

Trap-and-Emulate

- Trap to VMM on **sensitive** instructions
- VMM emulates the effect of these operations
- **VMM fools the guest OS into thinking that it runs at the highest privilege level** (or on a bare metal)

Pros: small overhead for non-sensitive inst.; no mod on gOS

Cons: large overhead for sensitive instructions
Recap - x86 difficulties (cpu)

- Not all sensitive instructions are privileged in x86.
- Hardware managed TLB gives Hypervisor no chance to intercept on TLB misses
- X86 has non-tagged TLB
  - Why this might be an obstacle for virtualization?
Recap - VMware **Full Virtualization** Solution

“VMware’s ESX Server dynamically rewrites portions of the hosted machine code to insert traps wherever VMM intervention might be required”

- CPU: binary translation
Recap - VMware Full Virtualization Solution

“ESX Server implements shadow versions of system structures such as page tables and maintains consistency with the virtual tables by trapping every update attempt — this approach has a high cost for update-intensive operations such as creating a new application process.”

- Memory: shadow page tables and pmap
Reasons against Full Virtualization

“These problems can be solved, but only at the cost of increased complexity and reduced performance.”

- Complexity
  - binary translation is very complicated
- Reduced performance
  - Management of shadow page table and pmap is expensive
  - Dynamic binary translation is expensive
Reasons against Full Virtualization

“In particular, there are situations in which it is desirable for the hosted operating systems to see real as well as virtual resources: providing both real and virtual time allows a guest OS to better support time-sensitive tasks, and to correctly handle TCP timeouts and RTT estimates, while exposing real machine addresses allows a guest OS to improve performance by using superpages or page coloring.”

- Detailed explanation on why these could be beneficial will be discussed later
ParaVirtualization

“We avoid the drawbacks of full virtualization by presenting a virtual machine abstraction that is similar but not identical to the underlying hardware”

- Tradeoff between improved performance with slight modifications to the guest operating system.
- Expose the existence of hypervisor to guest OS, rather than fool them.
## Cost of Porting an OS to Xen

<table>
<thead>
<tr>
<th>OS subsection</th>
<th># lines</th>
<th>Linux</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture-independent</td>
<td></td>
<td>78</td>
<td>1299</td>
</tr>
<tr>
<td>Virtual network driver</td>
<td></td>
<td>484</td>
<td>-</td>
</tr>
<tr>
<td>Virtual block-device driver</td>
<td></td>
<td>1070</td>
<td>-</td>
</tr>
<tr>
<td>Xen-specific (non-driver)</td>
<td></td>
<td>1363</td>
<td>3321</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>2995</strong></td>
<td><strong>4620</strong></td>
</tr>
</tbody>
</table>
| (Portion of total x86 code base  | 1.36%   |       | 0.04%|)

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).
ParaVirtualization Design Principles

1. Virtualize all architectural features required by existing standard ABIs. *Why is this important?*
   - ABI: application binary interface
2. Supporting full multi-application modern operating systems is important.
3. Paravirtualization is necessary to work on uncooperative machine architectures such as x86.
Xen
- First public version released in 2003
- Backing major cloud service e.g. AWS for many years
- Natively supported by Linux after kernel version 3.0 in 2011
  - KVM ported in 2006
- Not widely used now
Structure of XEN

- Control Plane Software
- User Software
- User Software
- User Software

- GuestOS (XenoLinux)
  - Xeno-Aware Device Drivers
- GuestOS (XenoLinux)
  - Xeno-Aware Device Drivers
- GuestOS (XenoBSD)
  - Xeno-Aware Device Drivers
- GuestOS (XenoXP)
  - Xeno-Aware Device Drivers

- Domain0 control interface
- virtual x86 CPU
- virtual phy mem
- virtual network
- virtual blockdev

- H/W (SMP x86, phy mem, enet, SCSI/IDE)
# Paravirtualized x86 Interface

<table>
<thead>
<tr>
<th>Memory Management</th>
<th>Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.</th>
</tr>
</thead>
<tbody>
<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 indirecting 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>

Details will be discussed in following sections.
Transparency vs. Optimization

Butler Lampson once gave a set of principles for system design. Among these, he gave two conflicting pieces of advice on the nature of implementations. He said,

“Keep secrets of the implementation. Secrets are assumptions about an implementation that client programs are not allowed to make... Obviously, it is easier to program and modify a system if its parts make fewer assumptions about each other.”

“One way to improve performance is to increase the number of assumptions that one part of a system makes about another; the additional assumptions often allow less work to be done, sometimes a lot less.”

That is, on the one hand we should **hide an implementation** for ease of development (transparency), and, on the other, we should **expose our implementations** for speed (optimization). *(cited from CSE221 homework)*
Xen chooses optimization while VMware chooses transparency
CPU Virtualization
## CPU Virtualization

### Memory Management
- **Segmentation**: Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.
- **Paging**: Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated contiguous machine pages.

### CPU
- **Protection**: Guest OS must run at a lower privilege level than Xen.
- **Exceptions**: Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.
- **System Calls**: 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.
- **Interrupts**: Hardware interrupts are replaced with a lightweight event system.
- **Time**: Each guest OS has a timer interface and is aware of both ‘real’ and ‘virtual’ time.

### Device I/O
- **Network, Disk, etc.**: Virtual devices are elegant and simple to access. Data is transferred using asynchronous I/O rings. An event mechanism replaces hardware interrupts for notifications.
Control Transfer: Hypercalls and Events

- Hypercall allows guest OS to perform a software trap to hypervisor for privileged operations
  - Similar to regular system call from user process to kernel

- Async events replaces device interrupts to perform notification from Xen to a domain
  - E.g. new data has been received from network
Protection (Privileged Instruction)

- Any guest OS attempt to directly execute a privileged instruction is failed by processor.
- All sensitive instructions are paravirtualized to be hypercall to Xen.

Graph credit to https://www.geeksforgeeks.org/virtualization-xen-paravirtualization/
Protection (Privileged Instruction)

<table>
<thead>
<tr>
<th>CPU Protection</th>
<th>Guest OS must run at a lower privilege level than Xen.</th>
</tr>
</thead>
</table>

Compare with VMware full virtualization on handling of non privileged sensitive instructions in x86

<table>
<thead>
<tr>
<th></th>
<th>Xen (ParaVirtualization)</th>
<th>VMware ESX (Full Virtualization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>Modification on guest OS to issue Hypercall for sensitive instructions</td>
<td>Dynamically rewrites portions of the hosted machine code to insert traps (binary translation)</td>
</tr>
<tr>
<td>pros</td>
<td>Lightweight and efficient</td>
<td>No need to port the guest OS</td>
</tr>
<tr>
<td>cons</td>
<td>Need to port OS to support Xen</td>
<td>Slow</td>
</tr>
</tbody>
</table>
Exceptions and System Calls

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

- Each guest OS registers an exception handler table with Xen for validation (no execution in ring 0)
- Some exceptions (e.g. system call) can be handled by guest OS directly through a installed fast handler
- Page fault must be handled by Xen for ring 0 privilege
## Exceptions and System Calls

### Without Virtualization

<table>
<thead>
<tr>
<th>Process</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. System call: Trap to OS</td>
<td>2. OS trap handler: Decode trap and execute appropriate syscall routine; When done: return from trap</td>
</tr>
<tr>
<td></td>
<td>3. Resume execution (@PC after trap)</td>
</tr>
</tbody>
</table>

*Figure B.2: System Call Flow Without Virtualization*

### With Virtualization

<table>
<thead>
<tr>
<th>Process</th>
<th>Operating System</th>
<th>VMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. System call: Trap to OS</td>
<td>2. Process trapped: Call OS trap handler (at reduced privilege)</td>
<td>4. OS tried return from trap: Do real return from trap</td>
</tr>
<tr>
<td></td>
<td>3. OS trap handler: Decode trap and execute syscall; When done: issue return-from-trap</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. Resume execution (@PC after trap)</td>
<td></td>
</tr>
</tbody>
</table>

*Figure B.3: System Call Flow with Virtualization*
Interrupts

Hardware interrupts are replaced with a lightweight event system.

- Hardware interrupts notify domains in the form of events
- Pending events stored in a per-domain bitmask
- Each guest OS specifies an event-callback handler which
  - Resets pending events set
  - Responds to events properly
- Event handling can be deferred by domain by setting a flag
  - Similar to disabling interrupts
- What's the benefit of using asynchronous event?
Time

- Real time: elapsed time in real world
- Virtual time: elapsed time only when the virtual machine is running

When could knowing real time be important?

When could knowing virtual time be important?
CPU Scheduling

Borrowed Virtual Time Scheduling algorithm (default):

- Support low-latency dispatch
  - latency sensitive processes (here domains) will be dispatched fast when events of them arrive
  - Ideal fair sharing of time slices is violated temporarily
- Can be configured to other scheduling algorithm on dom0
Performance

- Close to native setting (L-UP/L-SMP)
- Much better than VMW and UML

<table>
<thead>
<tr>
<th>Config</th>
<th>null call</th>
<th>l/O</th>
<th>stat</th>
<th>opensslct</th>
<th>sig inst</th>
<th>sig hndl</th>
<th>fork proc</th>
<th>exec sh 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>
</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>
</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>
</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>
</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>
</tr>
</tbody>
</table>

**Table 3: lmbench: Processes - times in μs**

<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><strong>1.97</strong></td>
<td><strong>2.22</strong></td>
<td><strong>2.67</strong></td>
<td><strong>3.07</strong></td>
<td><strong>28.7</strong></td>
<td><strong>7.08</strong></td>
<td><strong>39.4</strong></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>

**Table 4: lmbench: Context switching times in μs**
Memory Virtualization
Recap: Software-based Memory Virtualization

- Hypervisor will maintain a pmap to record the mapping from PPN to MPN
- When Guest OS tries to change page table, it will be trapped to hypervisor, and hypervisor will dynamically generate a Shadow Page Table
Recap: Hardware-assisted Memory Virtualization

- EPT is used to translate from PPN to MPN
- Hardware will first translate from VPN to PPN using guest page table, then use EPT to translate to MPN
Direct paging

- Guest OSes are responsible for the translation from PPN to MPN
  - Guest OS directly manages page tables
  - Page tables are mappings of VPN to MPN
- The top 64MB of each address space is reserved for Xen
  - This avoids the overhead of TLB flushes when entering or leaving hypervisors
Update page tables

- Page tables are read-only for Guest OSes
  - This is to prevent Guest OSes from making unacceptable changes
- Updates should be passed to Xen through a hypercall
  - Xen will validate this request before applied
    - No mapping to other guests’ pages
    - No write mapping to page table pages
  - Updates can be batched to reduce the number of hypercalls
    - This is particularly beneficial when creating new address spaces
- Guest OSes also can change the machine-to-physical table maintained by Xen through a hypercall
Maintain the information of pages

- Xen maintain a reference count, a type and owner for each page
- All pages are classified into the following types:
  - Page Directory (PD)
  - Page Table (PT)
  - Local Descriptor Table (LDT)
  - Global Descriptor Table (GDT)
  - Writable (RW)
- Types are used to validate the updates to page tables
- Types are also used to track which frames have already been validated for use in page tables
  - This obviates the need to validate the new page table on every context switch
- A page cannot be retasked until it is both unpinned and its reference count has reduced to zero
Page allocation and recycle

- The initial memory allocation of a domain is specified at the time of its creation and memory is statically partitioned between domains.
- If memory pressure of a domain increases, it may attempt to claim additional pages through a hypercall.
- If a domain wishes to save resources, perhaps to avoid incurring unnecessary costs, it can release pages back to Xen.
- A balloon driver is also used by Xen to recycle pages.
Page Faults

● Hypervisor will first check if this page fault happens in its own area
  ○ If so, hypervisor will handle it by itself
  ○ If not, hypervisor will transfer to page fault handler registered by Guest OS
● Can we use an unmodified page fault handler?
Page Faults

- Hypervisor will first check if this page fault happens in its own area
  - If so, hypervisor will handle it by itself
  - If not, hypervisor will transfer to page fault handler registered by Guest OS

- Can we use an unmodified page fault handler?
  - CR2 register can only be read in ring 0!

- Xen will create a copy of the exception stack frame on the guest OS stack, including the value of CR2 register
Full process of a page fault

- Application
- Page Fault
- Hypervisor Page Fault Handler
- Hypercall
- Hypercall Handler
- Application
- Guest OS Page Fault Handler
- Guest OS
Optimization: Writable Page Table

- Xen provides an alternative mode in which guest OSes can directly modify its page table
- When a guest tries to modify a page table page, Xen will temporarily mark this page writable, and disconnect it from the page table.
  - Why should Xen disconnect this page?
Optimization: Writable Page Table

- Xen provides an alternative mode in which guest OSes can directly modify its page table.
- When a guest tries to modify a page table page, Xen will temperately mark this page writable, and disconnect it from the page table.
  - Why should Xen disconnect this page?
- Xen will validate and reconnect the page when an address translation uses this page.
  - How does Xen capture this translation?
Discussion: SPT vs EPT vs Direct Paging
I/O Virtualization
Roadmap

- IO virtualization techniques used in other systems
- IO ring mechanism introduction
- Example for network receive rule
- Example for disk receive rule
- Advantages and disadvantages of such implementation
- Performance benchmark
- Related work: Usage of IO ring mechanism in other systems than Xen.
Recap: I/O virtualization techniques?
Recap: Other I/O virtualization techniques

1. Direct access to the device
2. Emulating the device
Recap: Other I/O virtualization techniques

3. Para-virtualization

4. Hardware assist
How does IO virtualization in Xen different than VMware?
VMware handles IO virtualization by interrupt and emulating devices.
Xen handle I/O virtualization using I/O ring and asynchronized event delivery mechanism

- Xen does not emulate hardware devices
- Exposes device abstractions for simplicity and performance.
- I/O data transferred to/from guest via Xen using shared-memory buffers
- Virtualized interrupts: lightweight event delivery mechanism from Xen-guest to xen
I/O ring mechanism introduction, with 4 descriptor pointers

- Ring is circular queue of descriptors
- Descriptors are allocated by guest OS
- Descriptors don’t directly contain I/O data
- Two pairs of producer/consumer pointers
- Domains place request
- Domain Advances request producer pointer
- Xen removes and handles them
- Xen advances request consumer pointer
- Zero copy transfer
Step 1, Domains place request
Domain Advances request producer pointer

Request Consumer
Private pointer in Xen

Request Producer
Shared pointer updated by guest OS

Response Producer
Shared pointer updated by Xen

Response Consumer
Private pointer in guest OS

- **Request queue** - Descriptors queued by the VM but not yet accepted by Xen
- **Outstanding descriptors** - Descriptor slots awaiting a response from Xen
- **Response queue** - Descriptors returned by Xen in response to serviced requests
- **Unused descriptors**
Step 2, Xen removes and handles them, asynchronously.

Request Consumer
Private pointer in Xen

Request Producer
Shared pointer updated by guest OS

Response Producer
Shared pointer updated by Xen

Response Consumer
Private pointer in guest OS

- Request queue - Descriptors queued by the VM but not yet accepted by Xen
- Outstanding descriptors - Descriptor slots awaiting a response from Xen
- Response queue - Descriptors returned by Xen in response to serviced requests
- Unused descriptors
Step 3, Xen advances request consumer pointer

- **Request Consumer**: Private pointer in Xen
- **Request Producer**: Shared pointer updated by guest OS
- **Response Producer**: Shared pointer updated by Xen
- **Response Consumer**: Private pointer in guest OS

- **Request queue**: Descriptors queued by the VM but not yet accepted by Xen
- **Outstanding descriptors**: Descriptor slots awaiting a response from Xen
- **Response queue**: Descriptors returned by Xen in response to serviced requests
- **Unused descriptors**
Step 4, Zero Copy to the domains

- **Request Consumer**
  - Private pointer in Xen
- **Request Producer**
  - Shared pointer updated by guest OS
- **Response Producer**
  - Shared pointer updated by Xen
- **Response Consumer**
  - Private pointer in guest OS

- **Request queue** - Descriptors queued by the VM but not yet accepted by Xen
- **Outstanding descriptors** - Descriptor slots awaiting a response from Xen
- **Response queue** - Descriptors returned by Xen in response to serviced requests
- **Unused descriptors**
Example - Network

- Xen models a virtual firewall-router (VFR) to which one or more VIFs of each domain connect
- Two I/O rings: one for send and another for receive
- Policy enforced by a special domain
  - Each direction also has rules of the form (<pattern>, <action>) that are inserted by domain 0 (management)
Example - Network

Packet transmission:
- Guest adds request to I/O ring
- Xen copies packet header, applies matching filter rules
- Round-robin packet scheduler

Packet reception:
- Xen applies pattern-matching rules to determine destination VIF
- Guest O/S required to provide PM for copying packets received
- If no receive frame is available, the packet is dropped
- Avoids Xen-guest copies;
Example - Disk

- Uses Split driver approach
- Front end, back end drivers
- Front end
  - Guest OSes use a simple generic driver per class
- Domain 0 (Xen) provides the actual driver per device
- Back end runs in own VM (domain 0)
Example - Disk

- Domain0 has access to physical disks
  - Currently(2003): SCSI and IDE
- All other domains are offered virtual block device (VBD) abstraction
- Created & configured by management software at domain0
- Accessed via I/O ring mechanism
- Possible reordering by Xen based on knowledge about disk layout
- Xen maintains translation tables for each VBD
- Used to map requests for VBD (ID,offset) to corresponding physical device and sector address
- Zero-copy data transfers take place using DMA between memory pages pinned by requesting domain
- Scheduling: batches of requests in round-robin fashion across domains
The file create and file delete time does not show significant improvement over other methods.

<table>
<thead>
<tr>
<th>Config</th>
<th>0K File create</th>
<th>0K File delete</th>
<th>10K File create</th>
<th>10K File delete</th>
<th>Mmap lat</th>
<th>Prot fault</th>
<th>Page fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-SMP</td>
<td>44.9</td>
<td>24.2</td>
<td>123</td>
<td>45.2</td>
<td>99.0</td>
<td>1.33</td>
<td>1.88</td>
</tr>
<tr>
<td>L-UP</td>
<td>32.1</td>
<td>6.08</td>
<td>66.0</td>
<td>12.5</td>
<td>68.0</td>
<td>1.06</td>
<td>1.42</td>
</tr>
<tr>
<td>Xen</td>
<td>32.5</td>
<td>5.86</td>
<td>68.2</td>
<td>13.6</td>
<td>139</td>
<td>1.40</td>
<td>2.73</td>
</tr>
<tr>
<td>VMW</td>
<td>35.0</td>
<td>9.3</td>
<td>85.6</td>
<td>21.4</td>
<td>620</td>
<td>7.53</td>
<td>12.4</td>
</tr>
<tr>
<td>UML</td>
<td>130</td>
<td>65.7</td>
<td>250</td>
<td>113</td>
<td>1k4</td>
<td>21.8</td>
<td>26.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>TCP MTU 1500</th>
<th>TCP MTU 500</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TX</td>
<td>RX</td>
<td>TX</td>
<td>RX</td>
</tr>
<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>VMW</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>
Performance Benchmark

Network benchmark shows significant improvements.

Smaller MTU -> More packets -> more interrupts -> more performance penalty.

<table>
<thead>
<tr>
<th>Config</th>
<th>0K File create</th>
<th>10K File create</th>
<th>Mmap lat</th>
<th>Prot fault</th>
<th>Page fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-SMP</td>
<td>44.9</td>
<td>24.2</td>
<td>123</td>
<td>45.2</td>
<td>99.0</td>
</tr>
<tr>
<td>L-UP</td>
<td>32.1</td>
<td>6.08</td>
<td>66.0</td>
<td>12.5</td>
<td>68.0</td>
</tr>
<tr>
<td>Xen</td>
<td>32.5</td>
<td>5.86</td>
<td>68.2</td>
<td>13.6</td>
<td>139</td>
</tr>
<tr>
<td>VMW</td>
<td>35.3</td>
<td>9.3</td>
<td>85.6</td>
<td>21.4</td>
<td>620</td>
</tr>
<tr>
<td>UML</td>
<td>130</td>
<td>65.7</td>
<td>250</td>
<td>113</td>
<td>1k4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCP MTU 1500</th>
<th>TX</th>
<th>RX</th>
<th>TCP MTU 500</th>
<th>TX</th>
<th>RX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>897</td>
<td>897</td>
<td>602</td>
<td>544</td>
<td></td>
</tr>
<tr>
<td>Xen</td>
<td>897 (-0%)</td>
<td>897 (-0%)</td>
<td>516 (-14%)</td>
<td>467 (-14%)</td>
<td></td>
</tr>
<tr>
<td>VMW</td>
<td>291 (-68%)</td>
<td>615 (-31%)</td>
<td>101 (-83%)</td>
<td>137 (-75%)</td>
<td></td>
</tr>
<tr>
<td>UML</td>
<td>165 (-82%)</td>
<td>203 (-77%)</td>
<td>61.1 (-90%)</td>
<td>91.4 (-83%)</td>
<td></td>
</tr>
</tbody>
</table>
Discussion: Pros and Cons of IO ring mechanism?
Discussion: Pros and Cons of IO ring mechanism

Advantages

Security
    More secure than grant direct access to the hardware
Performance improvement
    Since we use asynchronous event delivery mechanism and zero copy, the performance is significantly better than other IO virtualization technique.
## Discussion: Pros and Cons of IO ring mechanism

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Security</strong></td>
<td>Violate the fidelity rule.</td>
</tr>
<tr>
<td>More secure than grant direct access to the hardware</td>
<td>No guarantee delivery</td>
</tr>
<tr>
<td><strong>Performance improvement</strong></td>
<td>“If no frame is available, the packet is dropped.”If the descriptor buffer is filled up, there could be a chance we may lose the packets.</td>
</tr>
<tr>
<td>Since we use asynchronous event delivery mechanism and zero copy, the performance is significantly better than other IO virtualization technique.</td>
<td></td>
</tr>
</tbody>
</table>
Related work: Usage of IO ring mechanism in other systems

Linux!

QEMU-KVM v.s. Virtio v.s. Vhost-net v.s. Vhost-user

Virtio is a virtualization standard for network and disk device drivers where the guest's device driver just "knows" it is running in a virtual environment, and cooperates with the hypervisor. This enables guests to get high performance network and disk operations, and gives most of the performance benefits of paravirtualization.
Summary Discussion
What goals of Xen are not valid or less valid in today's cloud environments? Or, why Xen is not that popular now?

Use case of ParaVirtualization now?

https://dl.acm.org/authorize?N47257