Xen and the Art of Virtualization

CSE-291 (Cloud Computing)

Fall 2016

Why Virtualization?

- Share resources among many uses
- Allow heterogeneity in environments
- Allow differences in host and guest
- Provide a mechanism for migration, checkpointing, etc
  - Recovery, maintenance
- Provide for elasticity

Challenges to Virtualization

- Isolation
  - Performance w.r.t. resource sharing (CPU, memory, network, disk)
  - Protection w.r.t. resource use
- Support multiple operating systems
- Minimal performance penalty

Granularity

- Multiplex processes
  - Finely grained, but forces symmetric execution environment
- Multiplex OSes
  - Allows heterogeneous environments
  - High overhead of OS, runtime wastage and start-up latency
  -

Full Virtualization

- Can be expensive
- Time might not be a resource that it is desirable to virtualize
  - Timeouts, etc
  - Real network addresses allow better integration with real-world
- Much easier with hardware support

Paravirtualization

- “Good enough” virtualization
- Very similar to fully virtualized machine, but some differences that can be leveraged by knowledgeable OS
  - OS needs to be tweaked
  - Apps do not need to be changed

Xen Design Principles

1. Support for unmodified application binaries is essential, or users will not transition to Xen. Hence we must virtualize all architectural features required by existing standard ABIs.

2. Supporting full multi-application operating systems is important, as this allows complex server configurations to be virtualized within a single guest OS instance.

3. Paravirtualization is necessary to obtain high performance and strong resource isolation on uncooperative machine architectures such as x86.

4. Even on cooperative machine architectures, completely hiding the effects of resource virtualization from guest OSes risks both correctness and performance

Memory Management: Hardware Support

- Hardware Support: Software Managed TLB (Not available on x86)
  - Easiest option
  - Not available on x86
- Hardware Support: Tagged TLB (Not available on x86)
  - Use one tag per address space
  - Also a good option
- x86
  - Hardware managed page table
  - Flushed upon context switch

Virtualizing Memory Management On x86

- Implication: All valid page translations must be in page table
- Guest OSes allocate and manage page tables
- Xen needs to be mapped into all address spaces, so entering and leaving it doesn't require a flush or load.
  - But protected from Guest OS
- Guest OS allocates pages from own memory, but asks Xen to map it
  - Xen can validate/protect page tables
  - OS can batch updates for performance (amortize hypervisor overhead)

Virtualizing CPU: Privlege

- In model with only two privilege levels, hypervisor would need to be at OS's normal level of privilege
  - OS would need to run at app level, in its own context
  - Hypervisor would need to handle transitions, setting page tables, etc
    - Insert whole discussion about TLBs here
- X86 supports 4 privilege levels ("rings")
  - 0 normally for OS, 3 normally for apps
  - With Xen, Xen uses Ring-0, OS uses Ring-1
    - Isolates OS from Apps
    - Ensures only Xen can run privileged instructions

Virtualizing CPU: Privileged Instructions

- Paravirtualized Instructions
  - Validated and executed within Xen (Ring-0)
  - Processor fails or faults attempts other than from Ring-0
    - Guest OS can't directly execute
- Exceptions (Faults, Traps, etc)
  - Handled by Xen, which copies exception frame to Guest OS
  - Adjustment needed for page fault handler, since OS can't read address from CR2 register
    - Xen copies to an extended stack frame.
      - OS Needs to be modified to look there.

SysCalls and Page Faults

- Common enough to affect performance
- Syscalls
  - Guests register “fast interrupt handler” (top half)
  - Validated by Xen for install
- Page Faults
  - Can’t be optimized. Address is in CR2 and needs to be propagated by Xen

Device I/O

- Fully abstracted by Xen
- Simple interface
- Shared memory ring-buffers
- Callbacks via bit-maps replace interrupts
  - Xen flips bit and calling Guest OS event handler
    - Can also be blocked by OS

The Paravirtualized X86 Interface

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 discontiguous 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 indirecting 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.

Overall Structure

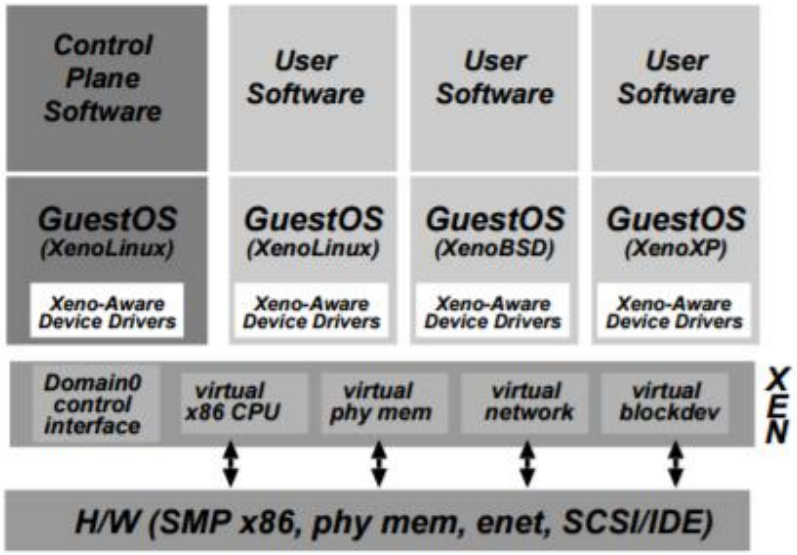

Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including *Domain0* running control software in a XenoLinux environment.

Control and Management

- Separation of Policy and Mechanism
- Xen exports interfaces
- Domain-0 created at boot time
  - Able to create new domains
  - Creates and deletes virtual network and block devices and associated access controls
  - Manages the resources associated with each domain
  - Guest OSes use interfaces to manage resources
  - Exports application management interface
    - Manage all of the resources

Hypercalls and Events

- Hypercall
  - Synchronous calls from domain into Xen
  - Basically a trap into the hypervisor, e.g. to request page table updates
- Events
  - Notifications to domain from Xen, e.g. to replace interrupts
  - Per domain bitmask is updated, followed by notification

Data Transfer: I/O Rings

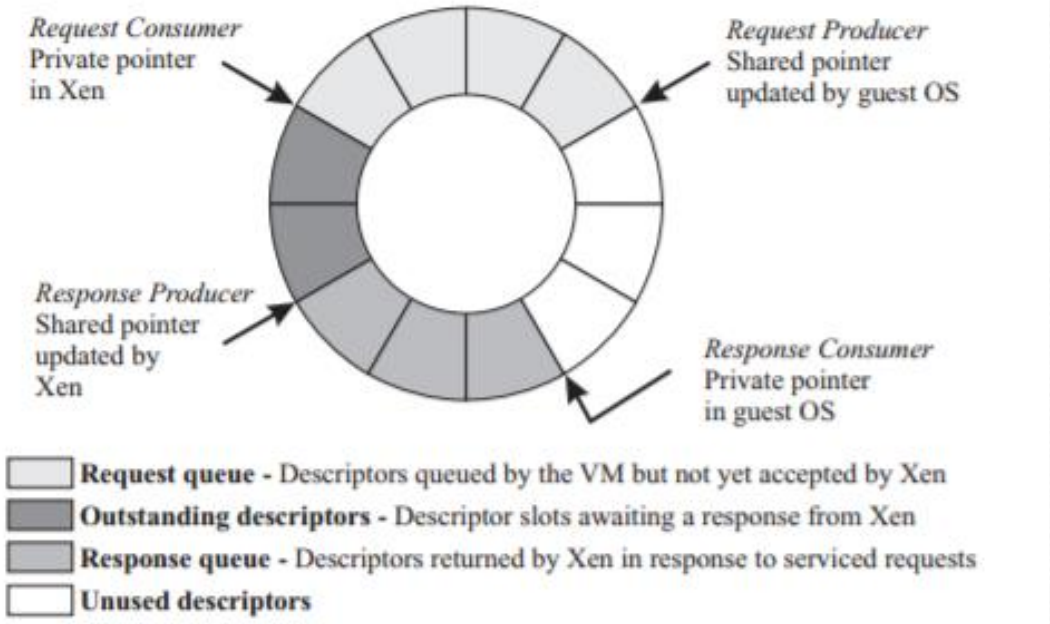

Figure 2: The structure of asynchronous I/O rings, which are used for data transfer between Xen and guest OSes.

- Circular ring
- Allocated by domain
- Accessible by Xen
- 2 pairs of producer-consumer pointers

CPU Scheduling

- Schedule domains via *Borrowed Virtual Time (BVT)*
  - Balances context switches, dispatch time, and fairness
  - Borrows from future time to allow warm threads to favor recently woken domains

Time and Timers

- Real time
  - Seconds since machine's boot to the accuracy of processor's cycle counter
- Virtual time
  - Advances only when VM is running
- Wall-clock time
  - Real time plus offset
  - Allows adjustment to real-time, without messing it up

Physical Memory

- Memory is statically partitioned to domains
- Memory can later be released
- Virtualized physical pages need to be mapped to physical pages for page tables
  - Xen provides this via a globally readable array
  - Can also be used to optimize for cache locality, etc.

Network

- Provides Virtual Firewall Router (VFR)
- Each domain has one or more Virtual Network Interfaces (VIF) attached to VFR
- Each VIF has two rings of buffer descriptors: transmit and receive
- Rules managed by Domain-0
- Transmit: Enqueue packet descriptor
  - Packet payload via scatter-gather DMA
  - Simple round-robin packet scheduler
- Receive: Exchange packet-buffer for empty frame on ring

Disk

- Only Domain-0 can access the actual disks
- Disks are represented as Virtual Block Devices (VBDs)
- Translation table from VBD offset to physical offset
- VBDs keep extents and access control information
- Guest scheduler may order requests going onto ring
- Xen can reorder after that, knowing more about storage
- Requests from competing domains are batched and round-robin scheduled
  - Fairness, but minimize overhead
  - Domains can specify reorder barriers to preserve higher-level semantics, e.g write-ahead log