Introduction to Xen

Yao-Min Chen

Outline

- Xen Overview
- Dom0 and DomU
- The role of Dom0
- Split device drivers
- Memory Management
- Live Migration

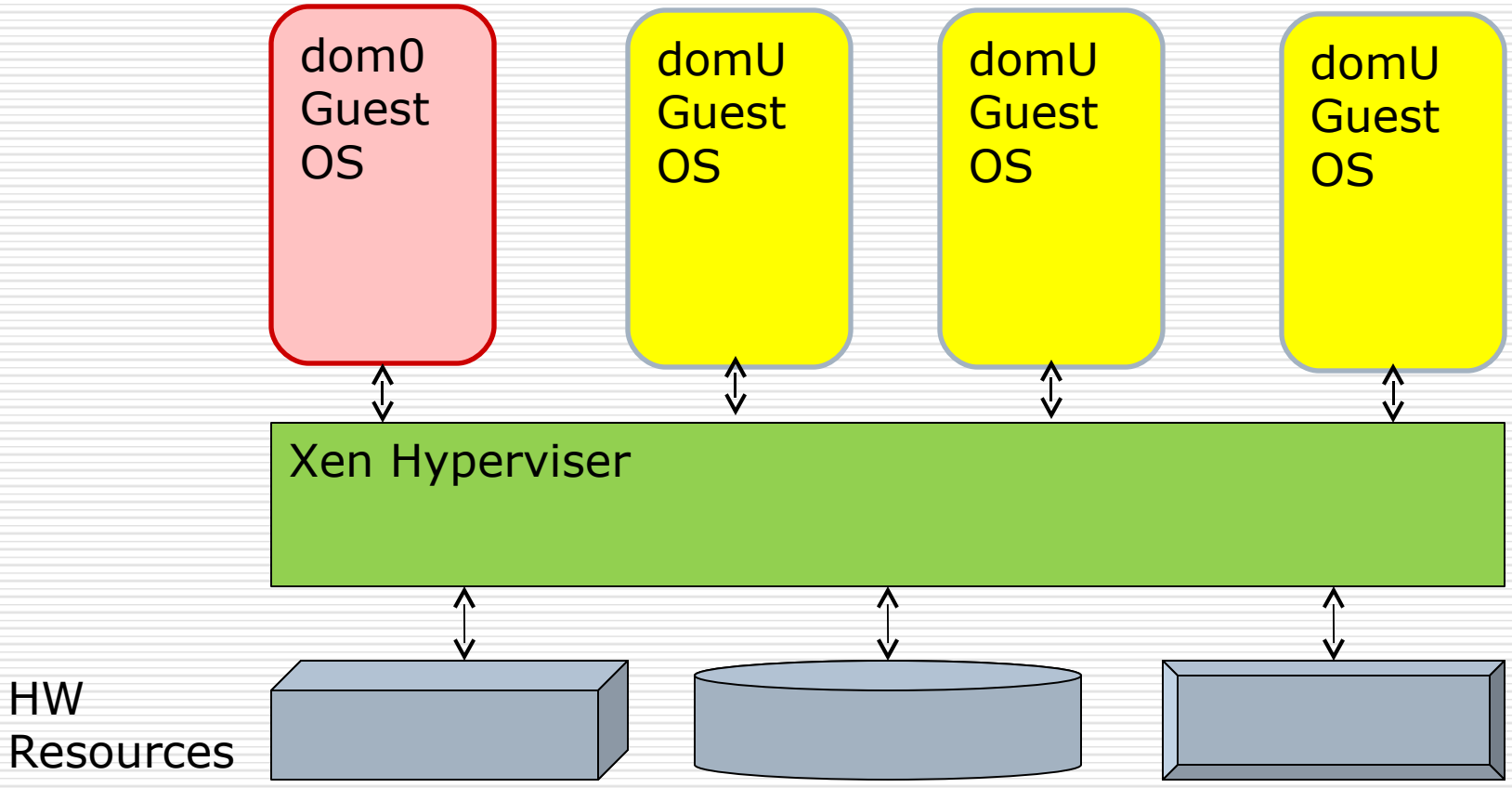
Xen Overview

- Xen – virtualization engine of many operating systems (Linux, Solaris/x64, BSD, etc)
- Open-source project, ported to many CPU architectures
  - X86, X86_64, Itanium, ARM
- Included in many Linux distributions
- Project founded in Cambridge U.
  - Commercialized by XenSource Inc., which was acquired Citrix
- Used by many companies entering Clouds

Xen Guests

- Domain 0 (dom0) Guest
  - Privileged – allowed to run privileged instructions and issue normal system calls
  - In x86 architecture, Ring 0 privilege level
  - Mostly Linux, but also Solaris and BSD
- Domain U (DomU) Guest
  - Unprivileged
  - Ring 3 privilege level

Xen Architecture



The role of dom0

- First guest to run when Xen starts
- Handling devices, including multiplexing of them for VMs
- Implementing half of the split device driver (the other half in domU)
- Handling administrative tasks (most through Python tools)
- Providing user interface to the hypervisor
  - *xend*: administrative interface to hypervisor
  - *xenstored*: back-end storage for XenStore

Split Device Driver

The minimalist approach by Xen

Xen Split Driver Model

- Most of kernel code is with drivers
- Xen leverages existing driver codes
- The real drivers are in dom0
  - Hence leveraging Linux device drivers
- Each guest OS (domU) only needs to implement emulated driver
  - Which forwards the driver calls to real driver in dom0

Xen Split Driver Model

- Top half – domU
- Bottom half – dom0
- Shared memory "grant tables" are used for data transfer between two halves of the drivers
- A common implementation is ring buffer built on top of grant table memory pages

Split Device Driver

Bottom Half @dom0

- Initializes a memory page and exports using grant table mechanism
- Advertise the grant reference via the XenStore
- Receives requests and write response into shared mem

Top Half @domU

- Find the bottom half shared memory page from XenStore
- Maps the page into its own address space
- Writes requests into the shared-memory page
- Receives responses

Xen Memory Management

Xen Memory Mgmt Services

- Allocating memory to guest OS
- Scrubbing free memory (ECC)
- Protecting guest OS from each other
- Enforcing typing rules (read versus write)
- Providing translation services between address spaces
  - Machine addresses
  - (Pseudo) Physical addresses
  - Virtual addresses

Three Paging Modes

- Direct Pagetables
  - For PV Guests
- Shadow Pagetables
  - For HVM Guests
- Hardware Assisted Pagetables

Terminology

- PFN: physical frame number
  - Guest's abstraction (imagination) for tracking/allocating RAM
  - Usually fairly contiguous ("linear space"), to be used by applications running on guest OS
- GFN: guest frame number
  - Guest's idea of what hardware addresses are
  - Used in guest page tables
- MFN: machine frame number
  - Actual hardware addresses

Types of Guests

- Para-Virtualized (PV) Guests
  - Aware of underlying Xen virtualization
  - Aware of machine frames
  - GFN == MFN
- Hardware Virtual Machine (HVM) Guests
  - Unaware of virtualization
  - Unaware of true machine frames
  - GFN != MFN

Direct Paging (PV Pagetables)

- GFN == MFN, so page tables can be used directly by the hardware
- Xen and Guests only concern with PFN-MFN mapping

Frame Mapping	Implementation
PFN -> MFN	PFN -> MFN table managed by the guest
MFN -> PFN	Shared MFN -> PFN table provided by Xen

Direct Paging Scenarios

- Application on Guest X wants to write to virtual address A
- Guest OS maps A to a pseudo physical address in frame Z
- Guest OS translate PFN Z to MFN Y
- Application data gets written on Frame Y
- Device writes data on MFN Y
- Xen determines the corresponding PFN Z belongs to Domain X
- Interrupt is delivered to Domain X
- Application in Domain X reads data from PFN Z

Shadow Pagetables

- For unmodified guest OS which
  - does not know about machine addresses
  - knows only pseudo physical addresses
- Xen marked the pagetables for guest as read-only
  - Xen keeps a copy of frames "shadowing" guest page-table frames (frames storing page tables)
- When guest OS access a frame,
  - It triggers modification to page table
  - Page fault is generated, and caught by Xen handler
  - Xen retrieves and update corresponding frame

Shadow Pagetables (Cont'd)

- GFN != MFN since guest OS is unaware of machine addresses

Frame Mapping	Implementation
GFN == PFN	Guest OS does not do machine address translation
PFN -> MFN	Mapping maintained by Xen; page fault trap at run time

Hardware Assisted Paging

- AMD-V
  - "Nested Paging" or Nested Page Tables (NPT)
- Intel VT
  - "Extended Page Tables (EPT)"
- Hardware assists in reducing memory access overhead due to virtualization
  - Earlier results with Xen did not show significant improvement
- Processors and Xen are both improving
  - Interesting to see the latest experimental data

Memory Management and Performance

- Memory management accounts for most of v14n complexity and overhead
- Hardware (processors) and software (hypervisor) co-development drives the direction in reducing the overhead
- The boundary of para-v14n and binary rewriting has become blurred
  - VMware added para-v14n
  - Xen has shadow paging support

I/O Virtualization

I/O Virtualization in Xen

- Similar problem as memory virtualization,
  - Physical memory (DMA address space) used by hardware device is different from what a guest OS sees in its memory
- Address translation is needed
  - Commonly done by IOMMU in modern CPUs

I/O Memory Management Unit (IOMMU)

- A specialized memory management unit that connects a DMA-capable I/O bus to the main memory.
- It maps device-visible virtual addresses (a.k.a. *device addresses* or *I/O addresses*) to machine addresses.
- In virtualization, it re-maps the addresses accessed by the hardware according to the same (or a compatible) translation table used by the virtual machine guest.

Live Migration

Move VMs from one machine to another

Live Migration of VMs

- In VMware, this feature is called vMotion
- Move running virtual machines from one physical server to another with no impact to end users
- For hardware maintenance, platform upgrade, high availability, and rebalancing the server load, to name a few

Live Migration Challenges

- Challenge: copying memory pages while some still being dirtied by apps
- Intelligent network support to insure live-migration traffic gets best QoS
  - Considering hardware acceleration
- What if we can't keep the original IP address?
  - Considering the case of migrating between VMs

Welcome Further Discussions

- Email: yaominchen@gmail.com
- Skype: yaominchen
- LinkedIn: Yao-Min Chen

Backup Slides

Useful Links

- Xen Source Code browser
  - http://lxr.xensource.com/lxr/source
- Hardware Assisted Paging virtualization performance tests
  - http://www.anandtech.com/show/3413

Paging with MMU

- Upon context switching (loading a process)
  - CR3 is updated with mm->pgd
  - TLB is flushed
- Memory request arrives and if there is a TLB miss
  - MMU accesses the Page Table Entry (PTE) using value in CR3
  - Address translation is cached in TLB
- When OS (Linux) removes ("flushes") an address translation entry, the corresponding TLB entry must be "invalidated"

TLB Flushing By OS

Scope	What it means	Linux example
TLB	Removes all mappings in TLB	void flush_tlb_all(void)
PGD	Remove all pages in address space	void flush_tlb_mm(struct mm_struct *mm)
Page Range	Remove some pages of the address space	void flush_tlb_range(struct mm_struct *mm, unsigned long start, unsigned long end);
Page	Removes a page	void flush_tlb_page(struct vm_area_struct *vma, unsigned long address);

Paging with Virtualization

- Global machine-to-physical table
  - Globally readable
  - Mapping between machine page frames to pseudo physical page frames
- Local physical-to-machine table
  - Per domain (guest OS)
  - Mapping between pseudo physical page frames to machine page frames

Protection Fault

- Page access exception.
- The page is not part of the program
  - so it is not mapped in program memory.
- Or, the program does not have sufficient privileges to read or write the page.
  - Supervisor mode versus user mode
  - Read only access versus read/write access
- HW interrupt is raised
  - Same interrupt as page fault, but different reason code.

X86 Protected Mode

- "Segmented" virtual address space
  - Code Segment
  - Stack Segment
  - Data Segment
- Local Descriptor Table (LDP)
  - Defines a set of segments accessible for the current process
- Global Descriptor Table (GDP)
  - Defines segments visible to all processes