CS236635
Network Functions Virtualization (NFV)

Class 3

Danny Raz
WWW http://webcourse.cs.technion.ac.il/236635

Instructor

Danny Raz (danny@cs.technion.ac.il)
Taub 512
Office hours: Monday 11:00-12:00

TA

Jalil Moraney (jalilm@cs.technion.ac.il)
Taub 533
Office hours: By appointment
## Schedule – tentative

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Class</th>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NC</td>
<td>NC</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>27.03.17</td>
<td>NFV introduction and Business Case</td>
<td>TA</td>
</tr>
<tr>
<td>3</td>
<td>03.04.17</td>
<td>Virtualization from Popek &amp; Goldberg to current Hypervisors</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>24.04.17</td>
<td>From the Hypervisor to the Cloud</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>08.05.17</td>
<td>Data Centers and the Cloud</td>
<td>Amazon cloud</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Google cloud</td>
</tr>
<tr>
<td>6</td>
<td>15.05.17</td>
<td>Modern Networking</td>
<td>Networking</td>
</tr>
<tr>
<td>7</td>
<td>22.05.17</td>
<td>Software Defined Networking</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>29.05.17</td>
<td>Datacenter Network</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Virtualization</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>05.06.17</td>
<td>Virtualizing Network Functions (VNF)</td>
<td>Project list</td>
</tr>
<tr>
<td>10</td>
<td>12.06.17</td>
<td>NFV – Process and Concepts</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>19.06.17</td>
<td>NFV Performance and Optimizations</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>26.06.17</td>
<td>NFV Placement</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>03.07.17</td>
<td>Student presentations and course recap</td>
<td>Cont. if needed</td>
</tr>
</tbody>
</table>
Questions?
Last week

P&G: Requirements vs. Theorem

• **Requirements:** what defines a Hypervisor
  - For something to be a Hypervisor, it must support
    1. Essentially identical environment
    2. Efficiency
    3. Hypervisor controls resources completely

• **Theorem:** condition that supports requirements
  - All sensitive operations are privileged (=trap in user mode)

“an efficient, isolated duplicate of the real machine”
Last week

Popek & Goldberg Theorem - Proof

- Network Function Virtualization
- TAU - 2014

Operating System

Non-sensitive

Sensitive

VM 1

VMM
Last week

Popek & Goldberg Theorem - Proof
Hypervisor Technology

- Hypervisors in practice – the x86 challenge
- Hypervisor taxonomy
- Hypervisor Optimization
  - Binary translation
  - Hardware Assistance - VT-x & VT-d [Intel]
  - Nested Hypervisors
- Leading Hypervisors Today
Terminology

• Host/Native OS
  – The OS sitting on the **physical** hardware

• Guest OS
  – The OS sitting on the **virtual** hardware, in the VM

• VM
  – Process that interacts with the VMM to simulate a physical machine internally

• VMM = Hypervisor = HV
  – Platform for supporting VMs
  – Different terms in different technologies – from here on will mostly use Hypervisor or HV

Network Function Virtualization - TAU - 2014
Terminology

- ISA (Instruction Set Architecture)
- API (Application Programe Interface)
- ABI (Application Binary Interface)

---

Network Function Virtualization - TAU - 2014
Hypervisor Types

Type 1
• Hypervisor installed on bare metal
  – No underlying OS

Type 2
• Hypervisor installed over existing OS

Which is better?

Hypervisor
Operating System
Hardware

Network Function Virtualization - TAU - 2014
Hypervisor Types

**Type 1**
- Hypervisor installed on **bare metal**
  - No underlying OS
  - Need to re-implement all OS functions
  - Not limited by OS design, less layers
- Example: VMWare ESXI

**Type 2**
- Hypervisor installed over existing OS
  - Have OS in place, only need to leverage for virtualization
- Example: KVM
Hypervisor Types

• Some Hypervisors do not fit nicely into the type definition
  – As time goes on, each architecture borrows from other designs
  – What constitutes an “operating system” can be disputed
• In the end, classification is just designed to help discuss the topic
• What really matters is...
  – Performance
  – Platform distribution
  – ...

Network Function Virtualization - TAU - 2014

13
x86?

• Intel CPU unit
  – Various versions ended with 86 – 80286, 80386...
  – Usually implied the 32-bit instruction set of the CPU
• Term commonly used to imply all the technology surrounding these systems
x86 virtualization challenge

• As mentioned in last lecture, virtualization became less popular with PCs becoming more available
  – No support built in to x86 architecture
  – No central location where change to virtualization could be made
• Must support existing HW design as-is (ISA)
• In x86 architecture:
  – Kernel mode (Ring 0)
    • CPU may perform any operation allowed by its architecture, including any instruction execution, IO operation, area of memory access, and so on.
    • Traditional OS kernel runs in Ring 0 mode.
  – User mode (Ring 1 ~ 3)
    • CPU can typically only execute a subset of those available instructions in kernel mode.
    • Traditional application runs in Ring 3 mode.
Virtualization Architecture

• The goal of virtualization is to **virtualize the hardware**
• Why not do full HW emulation?
• Full HW emulation would be very inefficient
  ➔ Not really a hypervisor, not efficient (remember P&G)
• Two basic traditional approaches:
  – Full virtualization
  – Para-virtualization
• Current industry trends (to be discussed later in the course)
  – containers
  – microkernels
  – AWS Lambda Functions
Traditional OS

- When application invoke a system call:
  - CPU will trap to interrupt handler vector in OS
  - CPU will switch to kernel mode (Ring 0) and execute OS instructions

- When hardware event:
  - Hardware will interrupt CPU execution, and jump to interrupt handler in OS

Source: “Lecture slides of “Virtual Machine” course (5200) in NCTU”
VMM and Guest OS

- System Call
  • CPU will trap to interrupt handler vector of VMM
  • VMM jump back into guest OS

- Hardware Interrupt
  • Hardware make CPU trap to interrupt handler of VMM
  • VMM jump to corresponding interrupt handler of guest OS

- Privilege Instruction
  • Running privilege instructions in guest OS will be trapped to VMM for instruction emulation
  • After emulation, VMM jump back to guest OS

Source: “Lecture slides of “Virtual Machine” course (5200) in NCTU”
Virtualization Approaches

- **Full virtualization**
  - Virtualize all hardware functionality – perfect software replica
  - But how to do efficiently? Remember P&G...

- **Para-virtualization**
  - Virtualized HW is similar but not identical, to actual HW
  - Requires collaboration with the virtualized OS (not P&G compliant!)
Virtualization Approaches

- **Full virtualization**
  - Virtualize all hardware functionality – perfect software replica
  - Basic approach – trap & emulate as foundation
  - But how to do efficiently? Remember P&G...

- **Pros**
  - Existing Guest-OSs can run on VM as-is
  - Guest OS unaware of virtualization

- **Cons**
  - Restricts optimization opportunities
Virtualization Approaches

**Para-virtualization**

- Virtualized HW is **similar but not identical** to actual HW
  - Non-virtualizable commands converted into **Hypercalls** to the Hypervisor
- Requires collaboration with the guest OS

**Pros**
- Not limited by host OS architecture, can integrate optimizations

**Cons**
- Guest OS must be modified from original
- Guest OS knows it is virtualized

Does not meet P&G virtualization req.!
Para-Virtualization

• Para-Virtualization implementation:
  – In para-virtualization technique, guest OS should be modified to prevent invoking critical instructions.
  – Instead of knowing nothing about hypervisor, guest OS will be aware of the existence of VMM, and collaborate with VMM smoothly.
  – VMM will provide the hyper-call interfaces, which will be the communication channel between guest and host.

Source: “Lecture slides of “Virtual Machine” course (5200) in NCTU”
We replace the instruction by a self-defined macro. The original instruction is the parameter of the macro. This macro would send a software interrupt to VMM. When receiving the SWI number 0x190, VMM has the knowledge that the next instruction is a instruction which should be emulated.
Binary Translation

• How can we achieve efficient full HW emulation?
• Major impact on performance: trapping
  – Trap = exception, disruption in program flow
  – Need to change context from VM to Hypervisor
Binary Translation

• To avoid trapping, (VMWare for example) applies binary translation
  – Hypervisor detects sensitive commands before they are activated by VM →
  – Translates them into non-sensitive commands that will yield the same result in the Guest HW
Static approach vs. Dynamic approach

– Static binary translation
  • The entire executable file is translated into an executable of the target architecture
  • This is very difficult to do correctly, since not all the code can be discovered by the translator

– Dynamic binary translation
  • Looks at a short sequence of code, typically on the order of a single basic block, translates it and caches the resulting sequence
  • Code is only translated as it is discovered and when possible, branch instructions are made to point to already translated and saved code

Source: “Lecture slides of “Virtual Machine” course (5200) in NCTU”
Hardware Solution

– Some trap types do not need the VMM involvement
  • For example, all system calls invoked by applications in Guest OS should be caught by Guest OS only. There is no need to trap to VMM and then forward it back to guest OS, which will introduce context switch overhead

– Some critical instructions should not be executed by guest OS
  • Although we make those critical instructions trap to VMM, VMM cannot identify whether this trapping action is caused by the emulation purpose or the real OS execution exception

Source: “Lecture slides of “Virtual Machine” course (5200) in NCTU”
Intel VT-x

• In order to straighten those problems out, Intel introduces one more operation mode of x86 architecture
  – VMX Root Operation (Root Mode)
    • All instructions in this mode are no different to traditional ones
    • All legacy software can run in this mode correctly.
    • VMM should run in this mode and control all system resources
  – VMX Non-Root Operation (Non-Root Mode)
    • All sensitive instructions in this mode are redefined
    • The sensitive instructions will trap to Root Mode
    • Guest OS should run in this mode and be fully virtualized through typical “trap and emulation model”

Source: “Lecture slides of “Virtual Machine” course (5200) in NCTU”
Intel VT-x

- VMM with VT-x:
  - System Call
    - CPU will directly trap to interrupt handler vector of guest OS
  - Hardware Interrupt
    - Still, hardware events need to be handled by VMM first
  - Sensitive Instruction
    - Instead of trap all privilege instructions, running guest OS in Non-root mode will trap sensitive instruction only

Source: “Lecture slides of “Virtual Machine” course (5200) in NCTU”
System State Management

- Binding virtual machine to virtual CPU
  - VCPU (Virtual CPU) contains two parts
    - VMCS maintains virtual system states, which are handled by hardware
    - Non-VMCS maintains other non-essential system information, which is handled by software
  - VMM needs to handle Non-VMCS part
Hardware acceleration – VT-d

• In a physical setup, virtual memory is mapped to physical memory.
• In a VM, the emulated physical memory needs also to be mapped to "real" physical memory.
• Finally, Guest OS thinks it has all memory, starting from "0”
• **Standard implementation**: shadow tables
  – Hypervisor stores map between guest physical and the VM process virtual
  – MMU then takes VM process virtual and maps to physical address
Hardware acceleration – VT-d

• VT-d offers EPT – Extended page tables
  – Direct memory translation from the guest physical address to the host address.
  – Speeds up operation since there is no need to manage shadow tables in the VMM
Virtualization Approaches

Full Virtualization

Para-Virtualization

Source: “understanding Full Virtualization, Paravirtualization and hardware assist”, white paper by VMWare
Leading Hypervisors Today

- **ESXI series**
  - By VMWare
- **XEN**
  - By Citrix
- **KVM (Kernel-based virtual machine)**
  - Part of recent RedHat distributions
- And many others...
  - VirtualBox, etc.

<table>
<thead>
<tr>
<th></th>
<th>Type 1</th>
<th>Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full</strong></td>
<td>VMWare</td>
<td>KVM</td>
</tr>
<tr>
<td><strong>Para</strong></td>
<td>ESXI</td>
<td>XEN</td>
</tr>
</tbody>
</table>
VMWare ESXI Series

• Type 1 full virtualization
• Part of an entire virtualization eco-system
• Costly solution
XEN by Citrix

• Hybrid Hypervisor
• Based on opensource project XENSource
• Currently managed by Citrix
KVM – Kernel-based Virtual Machine

• Paravirtualized Type-2 Hypervisor
• But full virtualization on x86 with hardware virtualization extensions (Intel VT-X)
• Uses as basis a HW emulator named QEMU
• Slowly gaining traction in the industry
  – Part of some Linux distributions
Nested Hypervisors

• A homework question...
• Is it also important in practice?
• **YES!** Why?
• Many operating systems are starting to include virtualization technology as part of the basics
  – Window 7 can virtualize Windows XP
  – Linux distributions sent out with KVM
• So, if we want to have a VM with Windows 7 or Linux, it must support virtualization within it
  – Nested virtualization
• Nested Hypervisors involve performance decrease
  – Main challenge: reduce the impact
• Several architectural solutions to this
  – Many involve direct access to HW when possible
From the Hypervisor to the Cloud

• So we have virtualized hardware...
From the Hypervisor to the Cloud

- So we have virtualized hardware...
- What happens when we have a lot of them?
From the Hypervisor to the Cloud

- So we have virtualized hardware...
- What happens when we have a lot of them?
From the Hypervisor to the Cloud

- So we have virtualized hardware...
- What happens when we have a lot of them?

Why did I want all these in the first place?
From the Hypervisor to the Cloud - Goals

- We want to run services
  - Compute – VM to run programs
  - Block Storage – attach to VM, store data from VM operation
  - Object Storage – upload content (e.g. Dropbox)
  - Run full applications
  - Connect to existing infrastructure
From the Hypervisor to the Cloud - Goals

• We need **supporting services**
  – Monitoring
  – Image management
  – Network management
  – Identity management (authentication)

• We also want to profit by **renting out these services**
  – What does this require?
Meet Amanda, Dana, Emma

- **Amanda**
  - Cloud owner (e.g. Amazon)

- **Dana**
  - Cloud customer (e.g. Netflix)

- **Emma**
  - Dana’s customer (e.g. You!)
The Cloud – User Experience

• Resource allocation via **self-service**
  – Dana should be able to manage her own resources **directly**
    • No need to call Amanda on the phone and ask for additional resources

• **Support for multi-tenancy**
  – Allow Dana to **reuse resources** for different Emmas
    • Not multi-instance!
  – But Emmas should experience as having independent resources
  – Effectively means strong account management capabilities
    • Think: What if Emma also has customers?
But wait – there’s more...

• Beyond the UI, there are many new and exciting questions to ask, regarding
  – Infrastructure Management
  – Application Management
  – New products and opportunities
The Cloud: Infrastructure Management

• **Resource allocation**
  – How to make sure we have enough resources for all customers?
  – How to prioritize when needed?

• How to support **customer QoS** (=quality of service)
  – Backups, network, etc.

• **Monitoring** resources
  – what and how

• **How to arrange physical servers**
  – Datacenter design
The Cloud: Application Management

How does the cloud change application management?
The Cloud: Application Management

• Application Management
  – Deploy anytime, anywhere
  – Scaling
  – Healing
  – Snapshots
  – Migration
  – ...

How does the cloud change application management?
The Cloud: Application Management

• **Scaling** is done to deal with changes in application load
  – Vertical scaling (up/down)
    • Adding/reducing resources (e.g. CPU, memory) to existing VM
  – Horizontal scaling (out/in)
    • Creating an additional copy of a VM as part of load-balancing effort

• **Healing** is when we deploy a new VM instance to replace an old one that died
The Cloud: Application Management

- **Snapshots** of a VM can be taken as backups in case of failure
- **Migration** is when we move a set of VMs from one datacenter to another
  - “Follow the sun”
    - e.g. Stock Exchange App
  - Disaster avoidance
    - Move VM to avoid impact of weather
  - Cloud-wide health and load balancing
    - Divide load to optimize power consumption, network bottlenecks, and more
The Cloud: New Opportunities

Which applications are most suitable for Cloud?
The Cloud: New Opportunities

• Application Types
  – Unexpected loads
    • startups
    • video servers
  – Critical infrastructure – survivability
    • Function of replication/healing
• High load
  – Genome project
The Cloud: New Opportunities

• The XaaS terminology
  – IaaS – Infrastructure as-a-Service
  – SaaS – Software as-a-Service
  – PaaS – Platform as-a-Service

• More of these are being developed over time
  – Some just as applications, e.g. Load Balancer as a service
Let’s go back to Cloud management

• The need of a **single interface**
  – Before we had many types of HW
  – Now we have many types of Hypervisor!
  – Nothing has changed... 😞

• We need
  – Single interface layer to manage all resources
  – Supporting self Service & Multi-Tenancy
  – Cost-effective for provider and customer
Questions?