The Cloud

COS 316: Principles of Computer System Design

Amit Levy & Jennifer Rexford
Layering: When it works we get

- **Modularity**
  - Application developer and ethernet developer can work independently

- **Portability**
  - The narrow waist of IP

- **Hide complexity with abstraction**
  - Simple applications on top of TCP

- **Re-use**
  - Many transport protocols on top of IP
Layering and Its Discontents

- How layering can *fail* and why abstraction can sometimes be *bad*
  - Poor choices of abstraction
  - Too much abstraction
- When to “pop open the hood”?
- Important, ubiquitous systems sometimes an accident of history
What Problem Does the Cloud Solve?
What Problem Does the Cloud Solve?

= 

$\ldots$
The Cloud Problem

Want to get resources in a datacenter, but:

- Relatively few resources (multiplexing)
- For a short period of time (burst scalability)
- Without having to trust my neighbors (strong isolation)
- For cheap

Why *datacenter* resources?

- Reliable (e.g. redundant power and network)
- Efficient (e.g. close to cheap power, economies of scale)
- Fast (e.g. high speed data links)
- …
What is The Cloud?
What is The Cloud?

- An abstraction of unbounded **compute** and **storage**
  - We will only focus on compute today
- A set of software **abstraction layers** that enable, variously:
  - Burst scalability
  - Resource multiplexing
  - Strong isolation
  - High utilization
  - Programmer convenience*
- As much an economically-drive organization of resources as an engineering one

*This is almost always better achieved with a library than an enforced layer*
## OK, But What is The Cloud?

<table>
<thead>
<tr>
<th>Portability</th>
<th>Flexibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure-as-a-Service (IaaS)</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>Platform-as-a-Service (PaaS)</td>
<td>Docker container</td>
</tr>
<tr>
<td>Software-as-a-Service (SaaS)</td>
<td>Web request handler in Go</td>
</tr>
<tr>
<td>User-facing application</td>
<td>Canvas, GitHub, Instagram, Turbo Tax, Etherpad…</td>
</tr>
</tbody>
</table>

**Infrastructure-as-a-Service (IaaS)**
- Ethernet
- CPU
- Block Device

**Platform-as-a-Service (PaaS)**
- TCP
- Process
- File system

**Software-as-a-Service (SaaS)**
- HTTP
- Language Function
- No Storage

**Serverless, or Function-as-a-Service (FaaS)**
- Web request handler in Go
- Docker container

**User-facing application**
- Virtual Machine
- Canvas, GitHub, Instagram, Turbo Tax, Etherpad…

**Language Function**
- No Storage

**Process**
- Virtual Machine

**File system**
- Virtual Machine
OK, But What is The Cloud?

- **SaaS**
- **IaaS**
- **PaaS**
- **FaaS**

- 1990s
- 2010s
- 2020s but also 2000s
- 2000s
From Virtual Machines to Cloud Computing
Operating Systems Have *Thick* Abstractions

**UNIX File System**

- Files divide disk into non-contiguous chunks.
- File system layers need to interpose between applications and disk.

**Sockets**

- Use transport-layer specific abstractions
  - E.g., allocate by tuple of (destination address, destination port, source port)
- Networking subsystem routes each packet, allocates new sockets.
Big abstractions & multicore

The year is [some year in the 1990s]...

CPUs have 1 core

Operating systems are monolithic and centralized

But...

Single-core CPU acceleration is slowing...

Your next computer will have many cores
Scalability: Disco (1997)

“Extensive modifications to the operating system are required to efficiently support scalable machines.”

“[W]e examine the problem of extending modern operating systems to run efficiently on large-scale shared-memory multiprocessors without a large implementation effort. [...] We use virtual machines to run multiple commodity operating systems on a scalable multiprocessor.”

“Disco: Running Commodity Operating Systems on Scalable Multiprocessors”
Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum

Ed, Scott & Mendel (along with Diane Green) founded VMWare a year later in October 1998
Scalability: Disco (1997)

From Edouard Bugnion’s job talk at EPFL in 2014:

“First, Disco ran commodity operating systems on scalable MIPS multiprocessors. [...] Second, VMware Workstation is a successful commercial product that allows...”

Work by a few grad students (along with Diane Green at Stanford in 1991)

Work by 10,000s of employees at a $70B Company

SAME!
Virtual Machines

Virtualization presents a physical machine as though many guest OSs had exclusive access
Isn’t a VMM just an Operating System?

- Yes!
- No: API is hardware resources (disk, network card), not abstract interfaces (file system, socket)
## Virtual Machines vs. Processes

<table>
<thead>
<tr>
<th></th>
<th>Virtual Machine Interface</th>
<th>Process Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network</strong></td>
<td>Network device (ethernet, WiFi, etc…)</td>
<td>TCP &amp; UDP sockets</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>Block device</td>
<td>File System</td>
</tr>
<tr>
<td><strong>Compute</strong></td>
<td>CPU</td>
<td>Unprivileged subset of CPU (x86/ARM/RISC-V…)</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>Virtual &amp; physical memory addresses</td>
<td>Virtual addresses only</td>
</tr>
</tbody>
</table>
Virtual Machines

Virtualization presents a physical machine as though many guest OSs had exclusive access

But How??
**Virtual Machines**

**Virtual Machine Monitor (VMM)** runs in supervisor mode (in place of kernel)

*OK... but how???
Trap-and-Emulate: System Calls

Assumes typical separation of applications and operating systems:

1. Applications interact with hardware by issuing system calls
2. System calls trapped by hardware which switches to OS context
3. OS uses *protected instructions* to access hardware directly

Virtual Machine Monitor “routes” system calls to appropriate guest OS

OS, running in process mode, causes trap through hardware on *protected instruction*
Virtual machine (CPU in user mode)

for (int i=0; i<100;i++){
    compute_stuff(...);
}
disk_write_syscall()
more_compute()

Guest OS

Virtual machine Monitor (CPU in supervisor mode)

HW traps to VMM

VMM forwards to guest kernel

Trap-and-Emulate: System Calls
Trap-and-Emulate: I/O

```c
handle_disk_syscall() {
    check_permissions();
    compute_stuff();
    data = ioout_read_block(some_block_number);
    ...
}
```

`ioout_read_block(real_block_number);`
What if we *can’t* trap-and-emulate?

x86 didn’t fault when some protected instructions were executed by user programs.

E.g. setting interrupt flags with Pop-Flags (POPFL) instruction silently fails

- **Binary translation**
  - Before running VM code for the first time, translate it to replace with explicit calls to VMM
  - Disco & early VMWare

- **Para-virtualization**
  - Modify guest OSs to detect if they are running inside a VM and use different instructions
  - Xen (2003) & most VMMs since

- **Modify the hardware**
  - Intel VT-X, APICV, VT-d, SR-IOV, GVT-d, and on and on… starting 2005
Flexibility: VMWare Workstation, Parallels

1999-~2006

Use virtual machines alongside physical machines

Run Windows apps on Linux, and Linux apps on Windows
Cloud Computing: Amazon EC2

“Before the advent of Amazon EC2, you had to buy or rent sufficient servers to cover your present needs, and you also had to be able to anticipate [and] forecast [...] for enough hardware to accommodate(sic) [...] growth as well as bursts of traffic [...]

With Amazon EC2, you don’t need to acquire hardware in advance of your needs. Instead, you simply turn up the dial, spawning more virtual CPUs, as your processing needs grow.”

-Amazon EC2 Announcement, August 25th 2006 Jeff Barr
Did VMMs solve single-server scalability?

- Yes! Can harness increasing number of cores to run more virtual machines!
- No: VMMs don’t help scale *individual* applications
- No: VMMs don’t mediate *sharing* between virtual machines

A decade later:
“An Analysis of Linux Scalability to Many Cores”
Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, and Nickolai Zeldovich

Careful adaptation of OS kernels broke the scalability barrier
Did VMMs solve The Cloud Problem?

- Yes! Can provision datacenter resources in small chunks, for hours at a time
- No: not small enough chunks! An hour is too long!
  - PaaS, FaaS to the rescue
- Yes: actually VMMs *can* be provisioned in smaller chunks for less time
  - “My VM is Lighter (and Safer) than your Container” Filipe Manco et al, SOSP 2017
  - Firecracker VM, 2018
- Yes/No: Strong isolation between VMs!
  - Well… maybe not… wait till next time
Take Aways

- Traditional *implementations* of OS abstractions poor fit for multicore
- Difficult to re-implement thick abstractions
- VMMs introduced a much simpler machine abstraction
  - Just run multiple OSs to “scale”
  - A decade to take advantage of increased density before OSs caught up
- Would we still build VMWare/EC2 today?
- Up next… what happens when annoying academics get their hands on the cloud?