CS370 Operating Systems
Colorado State University
Yashwant K Malaiya
Spring 2021 L26
Virtualization

Slides based on
• Text by Silberschatz, Galvin, Gagne
• Various sources
Kernel vs OS: OS = {Kernel, UI, libraries/binaries}

Virtualization:

• How to allocate vCPUs to virtual machines?
  – Guidelines/experience. pCPU:vCPU is often 1:1 to 1:3 or more. A VM may not use a CPU all the time.

• How much memory should be allocated per VM? Depends. 0.5-1GB min?

• Running a VM inside a VM?
  • Possible for modern CPUs. Restrictions possible for older.

• How can V machines run concurrently? How can two OSs run at the same time? Just like processes..
FAQ

HDFS

• Hadoop Distributed File System: Large storage, large files, distributed and replicated storage

• 64MB blocks saved as 4KB blocks of underlying FS (ext4 etc), likely on different nodes, allowing access in parallel.

• Why 3 copies? 1 replica: MTDL ≈ 1 year, 2 replicas ≈ 10 years, 3 replicas ≈ 100 years assuming independent failures
Updates

• Canvas, MS Teams
• cs370@cs.colostate.edu: TAs and instructors
• Finals: Sec 001 Th 5/13/21 2-4PM
  – Different for Sec 001 and Sec 002/801
  – comprehensive but questions will mostly be from the second half.
  – Canvas, Respondus Lockdown Browser, built-in calculator
  – 1 blank sheet, must be displayed before & after and destroyed on camera
  – Questions of various types
Updates

• Project:
  – Final report: 4/29/21, 2-columns, citations
    • Slides/Videos also needed on Project Slides channel
    • Peer reviews needed
  – Specific requirements for Option A and Option B
    • Option B: 15 minutes demos each team
      • Doodle Sign-up sheet available on MS Team for Mon, Wed
      • Demos using MS Team

• HW3 Review: Tuesday

• Course review: Tuesday, Thursday
How to cite references

Purpose of including citation:
• Giving credit to the authors
• Mention the authority behind the claim

Unacceptable examples:
• [https://dl.acm.org/citation.cfm?id=224066](https://dl.acm.org/citation.cfm?id=224066)
  – What is it about? Who wrote it? When? Why should I trust it?
• Anderson et al, Serverless network file systems, ACM
  – ACM is not a journal or conference. When was this work done?
• Examples: IEEE template
Wish things were back to normal ..
Course Survey

• Will be available on Canvas.
Implementation of VMMs

https://microkerneldude.files.wordpress.com/2012/01/type1-vs-2.png
Memory mapping:

- On a bare metal machine: OS uses page table/TLB to map Virtual page number (VPN) to Physical page number (PPN) (physical memory is shared). Each process has its own page table/TLB.
  
  - VPN -> PPN

- VMM: Real physical memory (*machine memory*) is shared by the OSs. Need to map PPN of each VM to MPN (Shadow page table)

  PPN -> MPN
Live Migration

- Migration from source VMM to target VMM
  - Source establishes a connection with the target
  - Target creates a new guest
  - Source sends all read-only memory pages to target
  - Source starts sending all read-write pages
  - Source VMM freezes guest, sends final stuff,
  - Once target acknowledge that guest running, source terminates guest.
Live Migration

• Migration from source VMM to target VMM
  – Source VMM establishes a connection with the target VMM
  – Target creates a new guest by creating a new VCPU, etc
  – Source sends all read-only memory pages to target
  – Source starts sending all read-write pages to the target, marking them as clean
    • repeats, as during that step some pages were modified by the guest and are now dirty.
  – Source VMM freezes guest, sends VCPU’s final state, other state details, final dirty pages, and tells target to start running the guest
    • Once target acknowledges that guest running, source terminates guest
VIRTUAL APPLIANCES: “shrink-wrapped” virtual machines

• Developer can construct a virtual machine with
  – required OS, compiler, libraries, and application code
  – Freeze them as a unit ... ready to run

• Customers get a complete working package

• Virtual appliances: “shrink-wrapped” virtual machines

• Amazon’s EC2 cloud offers many pre-packaged virtual appliances examples of *Software as a service*

• *Question: do we really have to include a whole kernel in a shrink wrapped VM?*
CS370 Operating Systems
Colorado State University
Yashwant K Malaiya

Data Centers & Cloud Computing

Slides based on
• Text by Silberschatz, Galvin, Gagne
• Various sources
Data Centers

• Large server and storage farms
  – 1000s-100,000 of servers
  – Many PBs of data

• Used by
  – Enterprises for server applications
  – Internet companies
  – Some of the biggest DCs are owned by Google, Facebook, etc

• Used for
  – Data processing
  – Web sites
  – Business apps
Data Center architecture

Traditional - static
- Applications run on physical servers
- System administrators monitor and manually manage servers
- Storage Array Networks (SAN) or Network Attached Storage (NAS) to hold data

Modern – dynamic with larger scale
- Run applications inside virtual machines
- Flexible mapping from virtual to physical resources
- Increased automation, larger scale
Data Center architecture

Giant warehouses with:

- Racks of servers
- Storage arrays
- Cooling infrastructure
- Power converters
- Backup generators

Or with containers

- Each container filled with thousands of servers
- Can easily add new containers
- “Plug and play”
- Pre-assembled, cheaper, easily expanded
Server Virtualization

Allows a server to be “sliced” into Virtual Machines

- VM has own OS/applications
- Rapidly adjust resource allocations
- VM migration within a LAN

• Virtual Servers
  - Consolidate servers
  - Faster deployment
  - Easier maintenance

• Virtual Desktops
  - Host employee desktops in VMs
  - Remote access with thin clients
  - Desktop is available anywhere
  - Easier to manage and maintain
Data Center Challenges

Resource management
- How to efficiently use server and storage resources?
- Many apps have variable, unpredictable workloads
- Want high performance and low cost
- Automated resource management
- Performance profiling and prediction

Energy Efficiency
- Servers consume huge amounts of energy
- Want to be “green”
- Want to save money
Data Center Challenges

Efficiency captured as *Power Usage Effectiveness*

• Ratio of IT Power / Total Power
• typical: 1.7, Google PUE ~ 1.1

Larger data centers can be cheaper to buy and run than smaller ones

- Lower prices for buying equipment in bulk
- Cheaper energy rates
- Automation allows small number of sys admins to manage thousands of servers
- General trend is towards larger mega data centers
- 100,000s of servers
- Has helped grow the popularity of cloud computing
## Economy of Scale

<table>
<thead>
<tr>
<th>Resource</th>
<th>Cost in Medium DC</th>
<th>Cost in Very Large DC</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU cycle cost</td>
<td>2 picocents</td>
<td>&lt; 0.5 picocents</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>$95 / Mbps / month</td>
<td>$13 / Mbps / month</td>
<td>7.1x</td>
</tr>
<tr>
<td>Storage</td>
<td>$2.20 / GB / month</td>
<td>$0.40 / GB / month</td>
<td>5.7x</td>
</tr>
<tr>
<td>Administration</td>
<td>≈140 servers/admin</td>
<td>&gt;1000 servers/admin</td>
<td>7.1x</td>
</tr>
</tbody>
</table>
Data Center Challenges

Reliability Challenges

Typical failures in a year of a Google data center:

• 20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
• 3 router failures (have to immediately pull traffic for an hour)
• 1000 individual machine failures
• thousands of hard drive failures

Capacity provisioning

User has a variable need for capacity. User can choose among

Fixed resources: Private data center
• Under-provisioning when demand is too high, or
• Provisioning for peak

Variable resources:
• Use more or less depending on demand
• Public Cloud has elastic capacity (i.e. way more than what the user needs)
• User can get exactly the capacity from the Cloud that is actually needed

Why does this work for the provider?
– Varying demand is statistically smoothed out over many users, their peaks may occur at different times
– Prices set low for low overall demand periods
Amazon EC2 Instance types

On-Demand instances
• Users that prefer the low cost and flexibility of Amazon EC2 without any up-front payment or long-term commitment
• Applications with short-term, spiky, or unpredictable workloads that cannot be interrupted

Spot Instances (cheap)
• request spare Amazon EC2 computing capacity for up to 90% off
• Applications that have flexible start and end times

Reserved Instances (expensive)
• Applications with steady state usage
• Applications that may require reserved capacity

Dedicated Hosts
• physical EC2 server dedicated for your use.
• server-bound software licenses, or meet compliance requirements
### Amazon EC2 Prices (samples from their site)

**General Purpose - Current Generation Region: US East (Ohio)**

<table>
<thead>
<tr>
<th>Instance</th>
<th>vCPU</th>
<th>ECU</th>
<th>Memory (GiB)</th>
<th>Instance Storage (GB)</th>
<th>Linux/UNIX Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2.nano</td>
<td>1</td>
<td>Variable</td>
<td>0.5</td>
<td>EBS Only</td>
<td>$0.0058 per Hour</td>
</tr>
<tr>
<td>t2.small</td>
<td>1</td>
<td>Variable</td>
<td>2</td>
<td>EBS Only</td>
<td>$0.023 per Hour</td>
</tr>
<tr>
<td>t2.medium</td>
<td>2</td>
<td>Variable</td>
<td>4</td>
<td>EBS Only</td>
<td>$0.0464 per Hour</td>
</tr>
<tr>
<td>m5.4xlarge</td>
<td>16</td>
<td>61</td>
<td>64</td>
<td>EBS Only</td>
<td>$0.768 per Hour</td>
</tr>
<tr>
<td>m4.16xlarge</td>
<td>64</td>
<td>188</td>
<td>256</td>
<td>EBS Only</td>
<td>$3.2 per Hour</td>
</tr>
</tbody>
</table>

ECU = EC2 Compute Unit (perf), EBS: elastic block store (storage), automatically replicated
The cloud Service Models

Service models

• IaaS: Infrastructure as a Service
  – infrastructure components traditionally present in an on-premises data center, including servers, storage and networking hardware
  – e.g., Amazon EC2, Microsoft Azure, Google Compute Engine

• PaaS: Platform as a Service
  – supplies an environment on which users can install applications and data sets
  – e.g., Google AppEngine, Heroku, Apache Stratos

• SaaS: Software as a Service
  – a software distribution model with provider hosted applications
  – Microsoft Office365, Amazon DynamoDB, Gmail
### The Service Models

<table>
<thead>
<tr>
<th></th>
<th>On-Premises</th>
<th>IaaS</th>
<th>PaaS</th>
<th>SaaS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Applications</td>
<td>Applications</td>
<td>Applications</td>
<td>Applications</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Runtime</td>
<td>Runtime</td>
<td>Runtime</td>
<td>Runtime</td>
<td>Runtime</td>
</tr>
<tr>
<td>Middleware</td>
<td>Middleware</td>
<td>Middleware</td>
<td>Middleware</td>
<td>Middleware</td>
</tr>
<tr>
<td>O/S</td>
<td>O/S</td>
<td>O/S</td>
<td>O/S</td>
<td>O/S</td>
</tr>
<tr>
<td>Virtualization</td>
<td>Virtualization</td>
<td>Virtualization</td>
<td>Virtualization</td>
<td>Virtualization</td>
</tr>
<tr>
<td>Servers</td>
<td>Servers</td>
<td>Servers</td>
<td>Servers</td>
<td>Servers</td>
</tr>
<tr>
<td>Storage</td>
<td>Storage</td>
<td>Storage</td>
<td>Storage</td>
<td>Storage</td>
</tr>
<tr>
<td>Networking</td>
<td>Networking</td>
<td>Networking</td>
<td>Networking</td>
<td>Networking</td>
</tr>
</tbody>
</table>

Cloud Management models

• **Public clouds**
  – Utility model
  – Shared hardware, no control of hardware,
  – Self-managed (e.g., AWS, Azure)

• **Private clouds:**
  – More isolated (secure?)
  – Federal compliance friendly
  – Customizable hardware and hardware sharing

• **Hybrid clouds:**
  – a mix of on-premises, private cloud and third-party, public cloud services.
  – Allows workloads to move between private and public clouds as computing needs and costs change.
Different Regions to Achieve HA

• AWS datacenters is divided into regions and zones, that aid in achieving availability and disaster recovery capability.

• Provide option to create point-in-time snapshots to back up and restore data to achieve DR capabilities.

• The snapshot copy feature allows you to copy data to a different AWS region.
  • This is very helpful if your current region is unreachable or there is a need to create an instance in another region
  • You can then make your application highly available by setting the failover to another region.
Different Regions to Achieve HA

Global Amazon Web Services (AWS) Infrastructure

- GovCloud (US ITAR Region) 2
- US West (Northern California) 3
- US East (Northern Virginia) 5
- Europe West (Dublin) 3
- Asia Pacific (Singapore) 2
- Asia Pacific (Tokyo) 2

# - Zones
- AWS Regions
- AWS Edge Locations (CloudFront & Route 53)
Containers

Slides based on
- Text by Silberschatz, Galvin, Gagne
- Various sources
Linux Containers and Docker

**Linux containers (LXC) are “lightweight” VMs**

**Comparison between LXC/docker and VM**

- Containers provide “OS-level Virtualization” vs “hardware level”.
- Containers can be deployed in seconds.
- Very little overhead during execution, just like Type 1.
Container technique

• Various technologies
  – Solaris Zones, BSD jails, LXC, Dockers etc.

• Linux kernel provides the “control groups” (cgroups) functionality for a set of processes
  – allows allocation and prioritization of resources (CPU, memory, block I/O, network, etc.) without the need for starting any VM
  – “namespace isolation” functionality
    • allows complete isolation of an applications' view of the operating environment including Process trees, networking, user IDs and mounted file systems.
What is a Container?

- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Includes specific Bins/Libs
- Works for all major Linux distributions
- Containers native to Windows Server 2016
- Docker: a popular container management service technology
## VMs vs Containers

<table>
<thead>
<tr>
<th>VMs</th>
<th>Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heavyweight: several GB</td>
<td>Lightweight: tens of MB</td>
</tr>
<tr>
<td>Limited performance</td>
<td>Native performance</td>
</tr>
<tr>
<td>Each VM runs in its own OS</td>
<td>All containers share the host OS</td>
</tr>
<tr>
<td>Hardware-level virtualization</td>
<td>OS virtualization</td>
</tr>
<tr>
<td>Startup time in minutes</td>
<td>Startup time in milliseconds</td>
</tr>
<tr>
<td>Allocates required memory</td>
<td>Requires less memory space</td>
</tr>
<tr>
<td>Fully isolated and hence more secure</td>
<td>Process-level isolation, possibly less secure</td>
</tr>
</tbody>
</table>
Some Docker vocabulary

**Docker Image**: analogous to executable
The basis of a Docker container. Represents a full application

**Docker Container**: analogous to a process
The standard unit in which the application service resides and executes

**Docker Engine**
Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

**Registry Service (Docker Hub (Public) or Docker Trusted Registry (Private))**
Cloud or server based storage and distribution service for your images

**DockerFile**: analogous to a makefile
The commands a user could call on the command line to assemble an image. Using docker build users can create an automated **build** that executes several command-line instructions in succession.
Using Docker: Build, Ship, Run Workflow

**BUILD**
Development Environments

**SHIP**
Create & Store Images

**RUN**
Deploy, Manage, Scale

Developers

IT Operations

Colorado State University
Docker Volumes

- Volumes mount a directory on the host into the container at a specific location

- Can be used to share (and persist) data between containers
  - Directory persists after the container is deleted
    - Unless you explicitly delete it

- Can be created in a Dockerfile or via CLI
Docker Compose: Multi Container Applications

- Build and run one container at a time
- Manually connect containers together
- Must be careful with dependencies and start up order

- Define multi container app in compose.yml file
- Single command to deploy entire app
- Handles container dependencies
- Works with Docker Swarm, Networking, Volumes, Universal Control Plane
Docker Compose: Multi Container Applications

version: '2'  # specify docker-compose version

# Define the services/containers to be run
services:
  angular:  # name of the first service
    build: client  # specify the directory of the Dockerfile
    ports:
      - "4200:4200"  # specify port forwarding

  express:  # name of the second service
    build: api  # specify the directory of the Dockerfile
    ports:
      - "3977:3977"  # specify ports forwarding

  database:  # name of the third service
    image: mongo  # specify image to build container from
    ports:
      - "27017:27017"  # specify port forwarding
Unique features

- Containers run in the user space
- Each container has its own: process space, network interface, booting mechanism with configuration
- Share kernel with the host
- Can be packaged as Docker images to provide microservices.
Monolithic architecture vs microservices
Microservices

eShopOnContainers reference application
(Development environment architecture)

Microservices Accessing the Shared Database

Each container is fully self-sufficient except that it uses a subset of the shared DB. A single DB subset can be accessed only by a dedicated container.
Microservices Characteristics

• Many smaller (fine grained), clearly scoped services
  – Single Responsibility Principle
  – Independently Managed

• Clear ownership for each service
  – Typically need/adopt the “DevOps” model

• 100s of MicroServices
  – Need a Service Metadata Registry (Discovery Service)

• May be replicated as needed

• A microservice can be updated without interruption
Microservices. Scalability

A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers

A microservices architecture puts each element of functionality into a separate service...

... and scales by distributing these services across servers, replicating as needed.
Reflecting on Part 1

• System structure and program compilation/execution

• Processes & Threads:
  – creation
  – scheduling
  – termination

• Inter-process communication
  – Synchronization
  – Deadlocks (included in Part 2)
Part 2

• We will review these on next Thursday.

• Virtual and physical address spaces
  – Pages and frames
    • Translation using page tables and TLBs
    • Effective access time
  – Virtual memory
    • Demand paging, page replacement algorithms
  – File systems
    • Disk organization, block allocation, scheduling
    • RAIDs
  – Virtual machines and containers
  – Data centers and cloud