CIS 455/555: Internet and Web Systems

Virtualization

September 20, 2021
Plan for today

- Programming abstractions for Web servers
- State, and where to keep it
- Virtualization basics
  - Emulation, binary translation
  - Hypervisors and virtual machines
  - Hardware support
- Containers
  - Union filesystems
  - Docker Hub
  - Mesos, Docker Swarm, Kubernetes
Idea: Build a “PC emulator”

- **Version 1: Interpreter for machine code**
  - Write a program that *emulates* the CPU and devices
  - Reads and interprets each instruction in the machine code for the OS, software, device drivers, etc.
  - E.g., Nintendo/Playstation emulators

- **Version 2: Binary translation**
  - Cross-compile the original machine code to the instruction set of the machine’s actual CPU
  - Can be static (compile-time) or dynamic (runtime)
  - Used in the past (e.g., Apple’s 68K→PowerPC, PowerPC→x86)
  - Today: Android ART, Microsoft’s fledgling x86-64-on-ARM
Virtualization

- What if we emulate a machine with the same instruction set?
  - No need to cross-compile! But...
  - Remaining challenge is to emulate “Ring 0” – privileged mechanisms for memory management, protection, etc.

- Originally done with clever software tricks
  - VMware etc.

- Today: Hardware support available
  - Example: Intel VT-x
  - Essentially provides a “ring -1” for the “hypervisor” or “VMM”
How do we manage VMs?

- **Language Level**
  - Application
  - Java Application
  - Virtualization Layer (JVM)
  - Operating System
  - x86 Architecture

- **Hosted**
  - Application
  - VM Operating System
  - Virtualization Layer
  - Host Operating System
  - x86 Architecture

- **Bare-metal**
  - App
  - Operating System
  - x86 Architecture
  - CPU
  - Memory
  - NIC
  - Disk
Virtual machine management

- What does the virtualization layer do?
  - Creates an emulated disk, emulated devices, ...
  - Executes the code quite efficiently
  - Multiple virtual machines (VMs) on a single physical one!
  - Can pause/migrate/reprovision, and the software in the VM won’t even notice it!
  - VMs can be scheduled preemptively to get (some) isolation
    - What are some of the challenges here?
Demultiplexing

- Network can make this transparent
  - A smart router could redirect incoming requests to the machine that is currently running the relevant server

- Can every server run every webapp?
  - Example: Solaris-based server with SPARC binaries
  - Need to emulate a full machine (with OS) for each server!
Problems with virtualization

- Some annoying aspects of virtualization:
  - Virtual disks are very bulky; lots of redundancy
    - Example: Two VMs have (almost) the same software and the same OS, only the data files differ
  - No good mechanisms for handling updates, composition, ...
  - Core parts of the OS are duplicated

- Result: Move towards containers
- Early example: Google’s “Borg” cluster
  - [https://research.google.com/pubs/pub43438.html](https://research.google.com/pubs/pub43438.html)
  - [https://github.com/google/lmctfy](https://github.com/google/lmctfy)
Recap: Virtualization

- Enables server consolidation
  - Single, shared pool of physical machines instead of dedicated machines for each service

- Can be implemented in different ways
  - Heavyweight: Interpretation
  - Slightly less heavyweight: Binary translation
  - Fairly efficient: Software tricks à la VMware
  - Efficient: Hardware support (VT-x)

- Some tradeoffs
  - Good: Strong isolation, works even across CPUs and Oses
  - Bad: Overhead
Plan for today

- Programming abstractions for Web servers
- State, and where to keep it
- Virtualization basics
  - Emulation, binary translation
  - Hypervisors and virtual machines
  - Hardware support
- Containers
  - Union filesystems
  - Docker Hub
  - Mesos, Docker Swarm, Kubernetes
Containers

- Let’s assume all servers target the same OS
  - Initially, Linux only

- Then we can use containers, which “virtualize” only the application-layer aspects
  - Kernel and devices are shared, but each container “sees” only its own resources and processes
  - They are also performance isolated
  - Containers can have a substructure (sub-containers)
Linux Containers (LXC)

- How do we isolate containers from each other?
  - Neither traditional process scheduling nor virtualization will help directly
  - Need new abstractions!

- Idea: **Namespace isolation**
  - Provide each container with a different view of the system

- Idea: **Control groups (cgroups)**
  - Enforce performance isolation
Namespaces

- Idea: restrict what a container can see!
  - Provide process level isolation of global resources
  - Processes have illusion they are the only processes in the system
- MNT: mount points, file systems (what files, dir are visible)?
- PID: what other processes are visible?
- NET: NICs, routing
- Users: what uid, gid are visible?
- chroot: change root directory
Control groups (cgroups)

- A collection of processes that share resources

- Resource isolation
  - what and how much can a container use?
  - Set upper bounds (limits) on resources that can be used
  - Fair sharing of certain resources

- Examples:
  - cpu: weighted proportional share of CPU for a group
  - cpuset: cores that a group can access
  - block io: weighted proportional block IO access
  - memory: max memory limit for a group
Containers virtualize at a higher layer

- Pro: Less overhead, less duplication
- Con: Kernel/hardware must be the same; isolation is harder
Impact on performance

- Containers have a lower overhead
  - Why?
- Speed is close to native speed
Docker: Union filesystems

- Let’s get rid of unnecessary duplication on the disk!

- Idea: Shared base image
  - Example: Ubuntu install
  - Base image is read-only (to maintain isolation between containers)

- Idea: Layer a second, writable FS “on top”
  - When a container changes its file system (including files from the base image), the changes/differences are stored in the writable FS and “overwrite” the base (incl. copy-on-write)
  - This is called a union filesystem
  - We can even ‘commit’ or ‘checkpoint’ the writable FS periodically and add another writable layer on top
A “Container Image” Composes Sub-Containers
Containers are based on a copy-on-write model with periodic “commits”
- ... just like a version-control repository! (Git, etc.)

Can we combine this with a GitHub-style model of cloning and committing?

Result: DockerHub
- ... and named, versioned images
- Many are using this to build reproducible computation!
Also Just Like git...

- Save only changed files
- Push or pull just small increment files
Containers in the real world

- Initially, containers were Linux-only
  - Now you can also have Windows or MacOS
  - Window and Mac support Linux containers, too

- Containers can be made visible to one another as machines on a (virtual) network
  - docker-compose (multiple containers together)
  - docker ... --link container
Containers and the Cloud

- Mesos, Docker Swarm, Kubernetes
  - Allow for more sophisticated deployment and management of many containers as “virtual clusters”
  - Can be mapped to real clusters, nodes can auto-restart, etc.

- Commercial cloud providers all have some analogous capability
  - Kubernetes is becoming the standard

For more information:
Plan for today

- State, and where to keep it
- Virtualization basics
- Containers
  - Union filesystems
  - Docker Hub
  - Mesos, Docker Swarm, Kubernetes
- Naming
How do we find things on the Internet?

- Generally, using one of three means:
  - **Addresses** or **locations**: we know where it is
    - Just like a physical address, we may still need a map!
    - In the Internet, addresses are typically IP addresses – the routers know the ‘map’
  - **Content-based addressing**: we know what it is
    - For instance, some keywords, hash of the content, ...
    - Basis of pub-sub systems, certain peer-to-peer architectures, indices (search engines like Google), NDN architecture
  - **Names**: we know what it is called
    - Best-known example on the Internet: DNS name
    - **Identifiers** are special names (1:1 mapping, no reuse)
    - Cell phone numbers, email addresses, etc. are becoming names
    - Need a way to map the name to an address/a location → Name resolution
Challenge: Find the 'right' architecture

- Should we use names or addresses?
- If names, what kind of names?
  - Need to define a namespace: Flat, hierarchical, ...
- How are names assigned?
  - Random, choose-your-own, explicit registration...
  - Do conflicts need to be handled? Who has authority?
- How do we resolve names?
  - Many different architectures: Flooding, centralized directory, hierarchical directory, decentralized directory ...
  - Is performance important? Can name mappings change? Should mappings depend on location, load, ...
- How can we secure the resolution system?
  - What threats exist? What defenses are available/appropriate?
Plan for today

- State, and where to keep it
- Virtualization basics
- Containers
  - Union filesystems
  - Docker Hub
  - Mesos, Docker Swarm, Kubernetes
- Naming
  - Flat naming
  - Attribute-based naming; LDAP
  - The Domain Name System (DNS)
Two simple architectures

- Directories can make lookup more efficient
  - Special data structures can further increase performance (inverted index etc. - stay tuned!)
- Directory can be distributed/hierarchical

Directories

- Directoryless
  - Gnutella, ARP
- Directory-based
  - LDAP, DNS, Napster 1.0
Case study: ARP

How does the switch know on which port it can reach a machine with a given IP address?

- Address resolution protocol (ARP); also uses flooding
- MACs are flat 48-bit IDs, but have some structure (OUI/NIC)
Case study: Gnutella

- Nodes choose their own file names
- Searches are flooded through the network
  - Node A wants a data item; it asks B and C
  - If B and C don’t have it, they ask their neighbors, etc.
  - Implications? Pros and Cons?
Case study: Napster 1.0 (ca. 2002)

- Hybrid of peer-to-peer storage with central directory showing what’s currently available
  - What are the trade-offs implicit in this model? Why did it fail?
What happened to Napster?

- May 1999: Napster founded by Shawn Fanning and Sean Parker
- Dec 1999: RIAA sues Napster for copyright infringement
- Apr 2000: Metallica & Dr. Dre sue Napster
- Jul 2000: U.S. District Judge Marylin Hall Patel orders Napster shut down
- Jul 2000: Appeals court stays injunction
- Feb 2001: Federal court rules Napster must stop trading in copyrighted material
- Jul 2001: Napster shuts down entire network
- Sep 2001: Case partially settled; Napster attempts to convert to a subscription system; filters copyrighted content
- Jun 2002: Napster files for Chapter 11 protection
  - Brand purchased by Best Buy in 2008, sold to Rhapsody in December 2011
  - Now a streaming music service
Recap: Flat naming

- There are several ways to look for things
  - Addresses/locations, names/identifiers, content

- How can we map names to locations?
  - Flooding: Robust but very expensive
  - Directories: Efficient but vulnerable
  - Examples: Gnutella, Napster, ARP (there are many others)